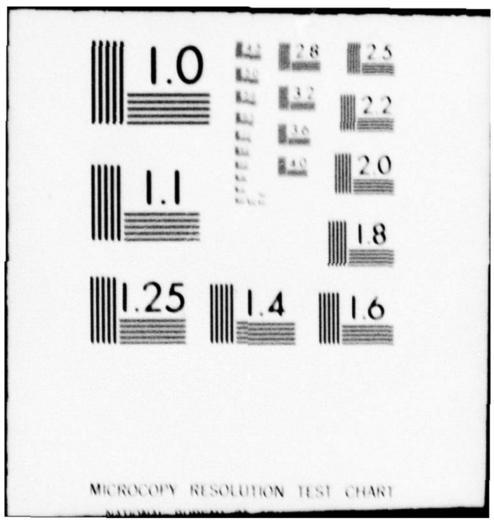
AD-A060 986 NEW MEXICO UNIV ALBUQUERQUE ERIC H WANG CIVIL ENGINE--ETC F/G 13/2
AIR FORCE REFUSE-COLLECTION SCHEDULING PROGRAM DESCRIPTION. VOL--ETC(U)
JUN 78 H J IUZZOLINO F29601-76-C-0015
UNCLASSIFIED CERF-EE-21 CEEDO-TR-78-23-VOL-3 NL

1 OF 3
AD
A060 986


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART

AD A060986

DDC FILE COPY

CEEDO-TR-78-23

AIR FORCE REFUSE-COLLECTION SCHEDULING PROGRAM DESCRIPTION VOLUME III: PROGRAM PHASE 3

LEVEL

HAROLD J. IUZZOLINO

ERIC H. WANG CIVIL ENGINEERING RESEARCH FACILITY
UNIVERSITY OF NEW MEXICO
BOX 25, UNIVERSITY STATION
ALBUQUERQUE, NEW MEXICO 87131

JUNE 1978

FINAL REPORT FOR PERIOD JANUARY 1976 – APRIL 1977

Approved for public release; distribution unlimited

CEEDO

CIVIL AND ENVIRONMENTAL ENGINEERING DEVELOPMENT OFFICE
(AIR FORCE SYSTEMS COMMAND)
TYNDALL AIR FORCE BASE
FLORIDA 32403

78 11 06 119

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 18 CEEDO-TR-78-23, Vol~~ume III~~ -3	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER 9
4. TITLE (and Subtitle) 6 AIR FORCE REFUSE-COLLECTION SCHEDULING PROGRAM DESCRIPTION. Volume III. Program PHASE3.		5. TYPE OF REPORT & PERIOD COVERED Final Report, Jan~~uary~~ 1976 ~~to~~ Apr~~il~~ 1977
		6. PERFORMING ORG. REPORT NUMBER 14 CERF-EE-21
7. AUTHOR(s) 10 Harold J. Iuzzolino		8. CONTRACT OR GRANT NUMBER(s) 15 F29601-76-C-0015
9. PERFORMING ORGANIZATION NAME AND ADDRESS Eric H. Wang Civil Engineering Research Facility, University of New Mexico, Box 25, University Station, Albuquerque, NM 87131		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS T.D. 4.03
11. CONTROLLING OFFICE NAME AND ADDRESS DET 1 (CEEDO) HQ ADTC Air Force Systems Command Tyndall Air Force Base, FL 32403		12. REPORT DATE 11 June 1978
		13. NUMBER OF PAGES 258
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) 12 257p.		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Available in DDC.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Minimum Number of Trips
Section Reassignments
Odd-Node Pairing
Path Optimization
Alternating Path

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes program PHASE3, the third of four programs in the Air Force Refuse-Collection Scheduling Program. Program logic, input, output, and limitations are presented in detail. A program listing and sample output are included.

78 11 06 119

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

400 976

LB

PREFACE

This report documents work performed during the period January 1976 through April 1977 by the University of New Mexico under Contract F29601-76-C-0015 with DET 1 (CEEDO) ADTC, Air Force Systems Command, Tyndall Air Force Base, Florida 32403. Captain Robert F. Olfenbuttel managed the program.

This volume, which documents program PHASE3, is the third of four volumes constituting the Air Force refuse-collection-scheduling program description. The algorithm used to obtain the minimum odd-node pairing in subroutines SOLV and NEXTN of program PHASE3 was suggested by Professor Donald R. Morrison. Shell's sorting algorithm is used in subroutines ISHLSRT and SHLSRT3. The remaining algorithms in program PHASE3 were developed by Harold J. Iuzzolino. Subroutine TRAVEL was coded by William Moore. The remaining subroutines and main program PHASE3 were coded by Harold J. Iuzzolino.

The report has been reviewed by the Information Officer and is releasable to the National Technical Information Service (NTIS). At NTIS it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

ROBERT F. OLFENBUTTEL, Capt, USAF, BSC Chief, Resources Conservation Br	PETER A. CROWLEY, Maj, USAF, BSC Director of Environics
EMIL C. FREIN, Maj, USAF Chief, Envmtl Engrg & Energy Rsch Division	JOSEPH S. PIZZUTO, Col, USAF, BSC Commander

(The reverse of this page is blank.)

TABLE OF CONTENTS

TABLE OF CONTENTS (Concl'd.)

LIST OF FIGURES

LIST OF TABLES

(The reverse of this page is blank.)

SECTION I
INTRODUCTION

1. OBJECTIVE

The primary objective of the Air Force Refuse-Collection Scheduling Program is to reduce refuse-collection costs. Program PHASE2 accomplishes the primary reduction, that of manpower costs, by assigning streets to collection sections in such a way as to produce the minimum number of trips. Program PHASE3 further reduces costs by generating efficient collection routes, which may also be minimum-length routes, for each section.

2. SCOPE

This section (Volume III) of the report describes the workings of the third program, PHASE3. A program overview is given, followed by a thorough description of the logic involved in route generation. A skeleton of the logic flow is provided. Input and output files are described. Program requirements and restrictions, error messages and error handling techniques, definitions of important variables, and an estimate of running time are also provided.

(The reverse of this page is blank.)

SECTION II
PROGRAM OVERVIEW

The purpose of program PHASE3 is to produce routes that fully describe the path each collection vehicle will travel. The distance traveled by each vehicle to service a section will be approximately the minimum possible distance, subject to limits on U-turns and the original selection of segments to be included in the section by program PHASE2. Two trips are produced for each section, one starting at the garage and one starting at the landfill, enabling program PHASE4 to further reduce the overall travel distance by an appropriate selection of trips.

Input to program PHASE3 consists of card data and four disk files. The card data must include a title card and a card giving the node numbers of the landfill and garage and may also include up to 500 cards that reassign segments to different sections. The disk files contain segment and node data for the entire map, a list of segments ordered by section, a list of vehicle capacities, and pointers to the first and last segments in each section.

The output produced by PHASE3 consists of printed output and one disk file. The disk file contains segment-by-segment descriptions of the paths servicing each section. These data will be used by program PHASE4 to generate the route maps and printed schedules. The printed output gives the initial and final segment assignments and vehicle capacities and a limited amount of data to be used for debugging problems that may arise.

The program consists of a main program named PHASE3 and 25 subroutines. The flow of control from one subprogram to another is displayed in Figure 1. Normal processing is shown. Within each subprogram, only the first call to each other subprogram is indicated. (Subroutine EXIT is a system subroutine and is not included in the description of program PHASE3.)

In its first stage, program PHASE3 reads the problem title from the first data card and the segments and their section assignments from disk files. The node data are also read from a disk file. The vehicle load is computed for

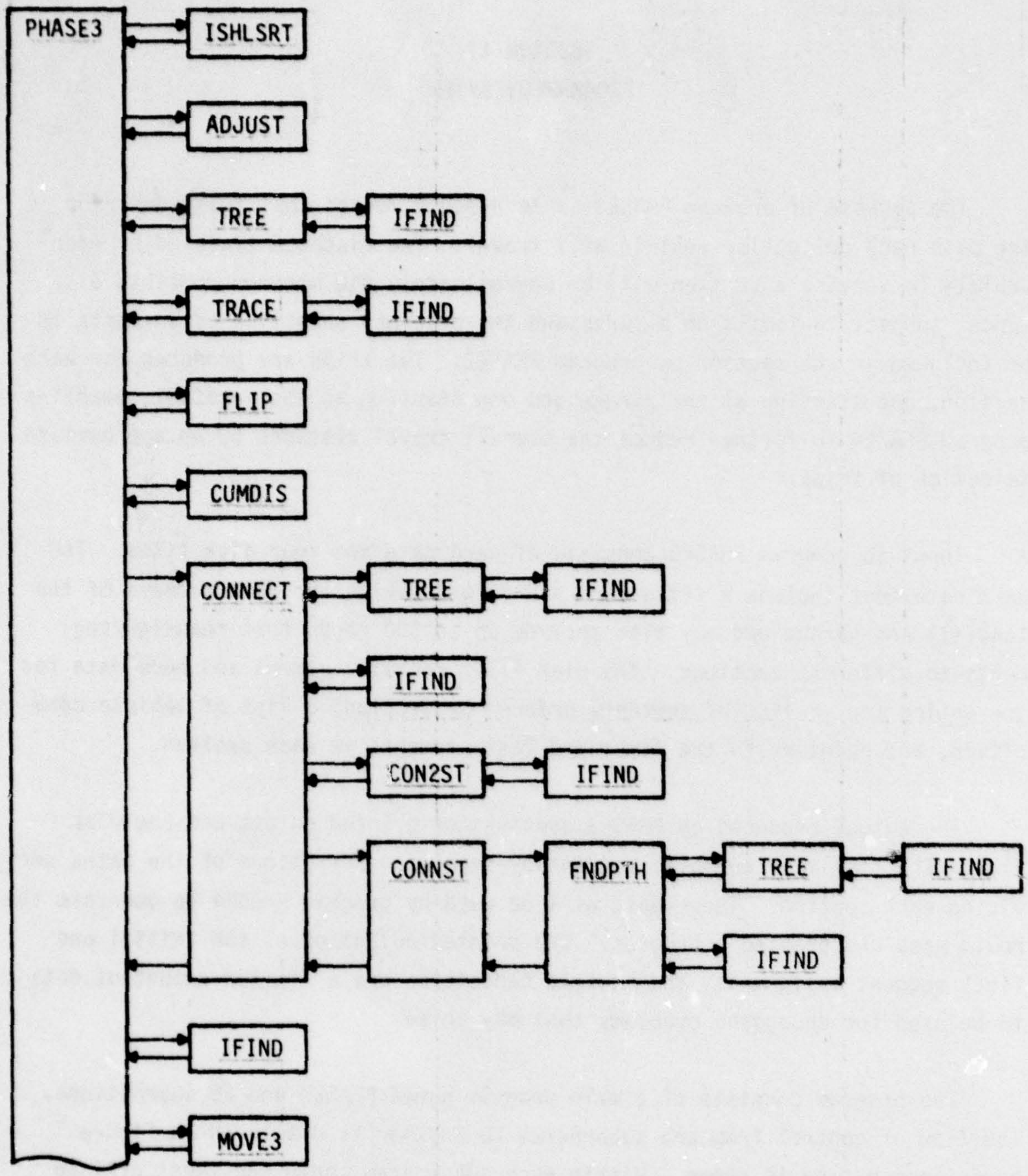


Figure 1. Control Relationships Among Subprograms

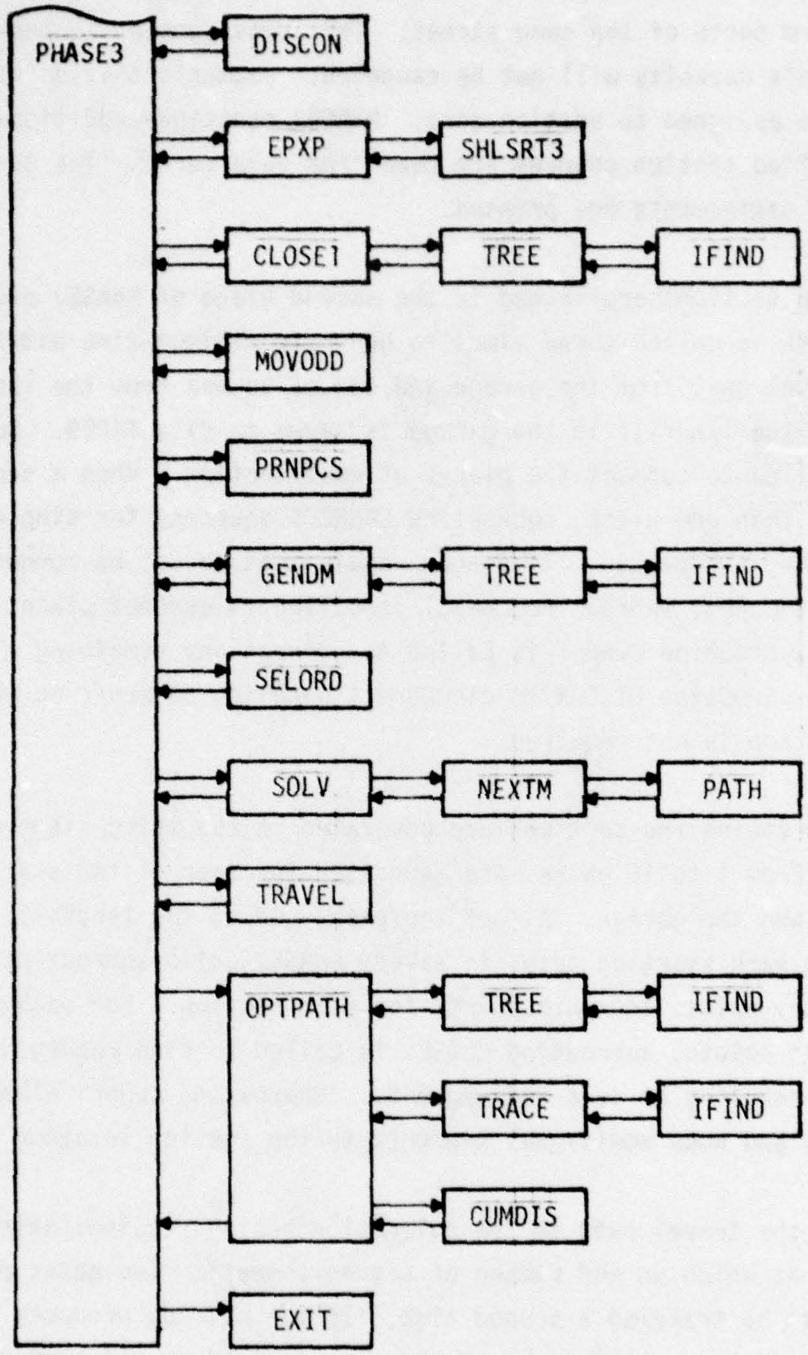


Figure 1. Control Relationships Among Subprograms (Concluded)

each section. Subroutine ADJUST is called to put into the same section all segments that form parts of the same street. This reassignment is performed only when a vehicle's capacity will not be exceeded. Segments that do not require collection are assigned to section zero. PHASE3 reassigns additional segments as user-specified section changes are read from data cards. The changes and final segment assignments are printed.

Connected sections are formed in the second stage of PHASE3 processing. Subroutine TREE is called three times to generate minimum-time paths to all nodes for travel away from the garage and travel to and from the landfill. The path from the landfill to the garage is saved on file TAPE9. Subroutine CONNECT is called to connect the pieces of each section. When a section consists of more than one piece, subroutine CONNECT searches for single-segment paths connecting the pieces. If pieces remain that cannot be connected by single-element paths, subroutine CON2ST is called to connect pieces with two-step paths. Subroutine CONNST is called to connect any remaining pieces. PHASE3 calls subroutine DISCON to disconnect dangling segments on which refuse-collection is not required.

Paths servicing the sections are generated in the third stage of PHASE3 processing. From 1 to 16 paths are generated for each of two starting points: the landfill and the garage. All of the paths end at the landfill. The shortest path from each starting point is saved. PHASE3 calls subroutine EPXP to select good entry points and exit points for each section. For each choice of entry and exit points, subroutine CLOSE1 is called to find return paths to the section from dangling or dead-end segments. Subroutine CLOSE1 allows U-turns in some cases and adds additional segments to the section in other cases.

Finding the travel path in the collection region requires pairing the odd nodes (nodes at which an odd number of segments meet). The paths pairing the odd nodes must be traveled a second time. If the pairing produces the minimum total pairing distance, and if collection is required on all segments in the region, then the travel path in the collection region will be of the minimum possible length. Some streets in the section, such as streets added by subroutine CONNECT to connect pieces of the section, may not require collection. These streets may cause the paths generated using a minimal odd-node pairing

scheme to be slightly longer than the minimum that could be obtained if streets not requiring collection were omitted.

PHASE3 calls subroutine GENDM to generate a matrix of distances between each pair of odd nodes. Subroutine SELORD is called to pair the odd nodes in a near-minimum manner. Subroutine SOLV is called to obtain an odd-node pairing with the minimum possible pairing length. PHASE3 computes the number of times each segment in the section must be traversed.

Subroutine TRAVEL is called to find, by trial and error, a path from the garage or landfill through the collection region and then to the landfill. Subroutine OPTPATH is called to perform some improvements on the path. The two shortest paths for each section, one starting at the garage and one starting at the landfill, are written to file TAPE9.

(The reverse of this page is blank.)

SECTION III
PROGRAM LOGIC

The logic for program PHASE3 is described from three viewpoints. The first description is task oriented. The second is data-storage oriented and includes discussions of the preparation of data for use by program PHASE4, the use of input data, and the data structures used in PHASE3. The third view describes each subroutine and the main program in terms of its purpose and the manipulations performed within it.

1. PROGRAM TASKS

The processing performed by PHASE3 can be grouped into four tasks: segment adjustments, section connections, odd-node pairing, and path generation and optimization. The data used during segment adjustments come from four files. The segment numbers on file TAPE4 are in order by section. File TAPE3 contains pointers marking the first and last segments in each section. All remaining data for the segments come from file TAPE1. Node data from file TAPE2 are also needed during the program-initiated section reassignments of the segments. The number of segments meeting at each node (the order of the node) is computed for each node from the packed word of bounding-segment numbers.

The program-initiated changes are performed by subroutine ADJUST. This subroutine examines the segments bounding each order-two node. If both of the segments contain refuse and each is serviced in a different section, the subroutine changes the section assignment of one of the segments (if the change will not cause the vehicle capacity to be exceeded). The reassignments are printed by the subroutine as they are made. The amount of refuse in each section is recomputed and printed after control returns to the main program.

The user-initiated section reassignments are controlled by data cards read by the main program. The segment number and its new section assignment are read from a data card. The segment is reassigned, and the corresponding

refuse quantity is transferred from the old section to the new section. The program makes no attempt to keep the refuse quantity in any section from exceeding the vehicle capacity; this is the user's responsibility. A message is printed describing each segment reassignment. The final refuse quantities for the sections are printed.

The first step in the formation of connected sections is to verify that each node is connected to the landfill and the garage. Subroutine TREE is called three times to generate paths from the landfill to each node and from the garage to each node. The paths between node and landfill are generated twice, once for travel toward the landfill and once for travel away from the landfill. The paths are built to minimize travel time. The path information consists of an array that gives the line number of the next node on the path to the landfill or garage and an array that gives the next segment in the path to the landfill or garage. If, at any node other than that of the landfill or garage, the line number of the next node in the path is zero or the number of the next segment in the path is zero, then that node is not connected to the landfill or garage. Error messages are printed when unconnected nodes are found.

Subroutine CONNECT is called once for each section to connect separate pieces of the section. The subroutine examines each segment in the map description until it finds a segment that is part of the desired section. The starting node of this segment is used as the starting point of a minimum-distance tree, but only segments within the section are used in building the paths. The remaining segments in the map description are examined. If a segment is in the section, but one of its nodes is not connected to the path, a separate piece of the section has been found. Paths are built from nodes in each subsequent piece to find all of the segments in each piece. When the segments have been marked according to the piece in which they occur, subroutine CONNECT adds to the section, for travel purposes only, each single segment that connects two pieces of the section.

If the section still consists of more than one piece, subroutine CON2ST is called to find the best two-step paths between the remaining pieces. If the section then still has more than one piece, subroutine CONNST is called to find

the shortest path connecting two pieces. These pieces are connected. If more than one piece yet remains, the next shortest path between two pieces is found. The process continues until all pieces of the section have been connected.

The next step in the formation of connected sections is the deletion of dangling or dead-end segments on which no collection is required. The main program calls subroutine DISCON to disconnect these segments.

The path through the collection region usually requires that some streets be traversed more than once. The path must use two segments, or one segment twice, each time it crosses a node. At an even-order node, each street can be traversed once. At an odd-order node, one street must be traversed a second time. Thus, the streets that must be traveled more than once will connect pairs of odd nodes.

The odd-node pairing is performed in two steps: generation of an initial pairing, and iterative improvement to obtain the pairing with the minimum total paired length. If collection is required on all streets in the collection region, the final odd-node pairing will produce the shortest possible travel distance in the region. The presence of streets not requiring collection, however, may cause the travel distance to exceed the minimum. Attempts to remove these streets will be described in the discussion of path optimization.

Subroutine EPXP is called by PHASE3 to select possible entry and exit nodes. The order of each node located where the travel path enters and leaves the collection region is increased by one. Odd-node pairing and path generation are performed for each combination of entry and exit nodes because the choice of these nodes can change the number of odd-node pairs and the total travel distance in the region. The path with the shortest total distance will be used. Subroutine EPXP chooses as entry points the nodes closest to the landfill or garage, provided that the path from these nodes to the landfill or garage includes no other entry-point nodes. The exit nodes are selected in a similar manner, using paths from the landfill to the section.

Since each dead-end or dangling segment has a node of order one, a return path to the section from that node is required. Rather than include the generation of these paths as part of the odd-node pairing, subroutine CLOSE1 is

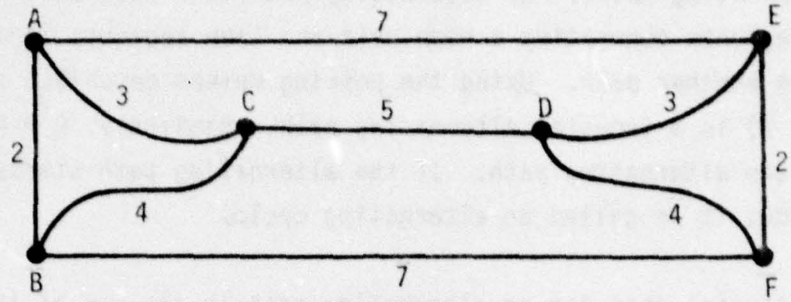
called by PHASE3 to generate the return paths. The subroutine may add additional nodes and segments to the section to provide a return path, or it may repeat the segment, creating a U-turn. The U-turn is used when the shortest alternate path traverses more than 32 segments.

Both the initial odd-node pairing and the iterative improvement require a matrix of distances from each odd node to each other odd node. The distance matrix is generated by subroutine GENDM. This subroutine uses subroutine TREE to build paths starting at each odd node. The distances to the other odd nodes are extracted from an array returned by subroutine TREE.

Subroutine SELORD is called by PHASE3 to generate an initial pairing of odd nodes. Two pairing methods are coded in the subroutine. The fifth argument is set to -1 to select one method, or to +1 to select the other. The first method examines the distance from each node to its nearest neighbor. The node having the longest such distance is paired with its nearest neighbor. If the nearest neighbor of any unpaired node has been used as part of an odd-node pair, a new near neighbor is found for the unpaired node. The nodes with the next longest near-neighbor distance are then paired, and the procedure is repeated until all nodes are paired. The second pairing method pairs nodes starting with the shortest near-neighbor separation. Again, the procedure is repeated with the remaining unpaired nodes until all are paired.

Since PHASE3 calls subroutine SELORD with a fifth argument of -1, the first method is always used. The procedure is illustrated by the network shown in Figure 2(a). The six odd nodes are designated by the letters A through E. The lengths of the segments are appended near the middle of the segments. The initial selection of near neighbors is shown in Figure 2(b). The largest separation has length 3 and is first encountered with nodes C and A. When nodes A and C are paired, node B loses its nearest neighbor. A new near neighbor, F, is found for node B. The resulting status of the unpaired nodes and their neighbors is shown in Figure 2(c). Nodes B and F now have the largest near-neighbor separation, so they are paired. The remaining nodes, D and E, must now be paired. The total length of the segments connecting the three pairs (A-C, B-F, and D-E) is 13.

(a) Sample Network



Notes: (1) Letters indicate odd nodes.
(2) Numerals indicate segment lengths.

(b) Node Neighbors Before First Pairing

Node	Neighbor	Separation
A	B	2
B	A	2
C	A	3
D	E	3
E	F	2
F	E	2

(c) Node Neighbors Before Second Pairing

Node	Neighbor	Separation
B	F	7
D	E	3
E	F	2
F	E	2

Figure 2. Pairing of Odd Nodes by Subroutine SELORD

To obtain pairs having a minimum total paired length, it is necessary to examine alternating paths. An alternating path is a path that alternately traverses segments connecting a node pair and then segments leading to a node belonging to another pair. Using the pairing method described above, A → C → D (Figure 2) is a two-step alternating path. Similarly, C → A → B → F → E is a four-step alternating path. If the alternating path starts and stops on the same node, it is called an alternating cycle.

The potential gain for an alternating path is the sum of the lengths of the segments connecting paired nodes minus the sum of the lengths of the segments connecting different pairs. All connecting lengths are taken along the shortest path. For example, A → C → D has a potential gain of -2, while C → A → B → F → E has a potential gain of 6. The alternating cycle C → A → B → F → E → D → C has the following potential gain: 3 - 2 + 7 - 2 + 3 - 5 = 4. A cycle with positive potential gain is called a profitable cycle. When a pairing is found that has no profitable cycles, that pairing has the minimum total pairing length.

The improvement of the odd-node pairing starts with the formation of a matrix of potential gains for two-step alternating paths. Let $PG_n(I,J)$ be the largest potential gain, or profit, of any path from I to J found while computing the N-step potential gain matrix. Paths that end at the partner of the starting node are assigned large negative potential gains (-1000). Two-step paths that start and end on the same node are assigned potential gains of 0. The shortest pair-connecting path may traverse the segments connecting a pair, but is still treated as an unpaired length. For example, A to E is A → C (paired) → E via D (pairs ignored) for a potential gain (PG) of 3 - (5 + 3) = -5. The two-step alternating path from C to D is C → A (paired) → D via C (unpaired). The PG_2 matrix for Figure 2 is shown in Table 1.

The PG_2 matrix is formed in subroutine SOLV. Subroutine NEXTM is called by subroutine SOLV to double the number of steps in the PG matrix. The old matrix is not destroyed by the new matrix. Subroutine NEXTM computes the diagonal entries (cycles) first. The entry from node I to node I is evaluated using the approach shown in the following equation.

TABLE 1. TWO-STEP PROFIT MATRIX

START \ STOP	A	B	C	D	E	F
A	0	-1	-1000	-2	-5	-6
B	-2	0	-2	3	5	-1000
C	-1000	1	0	-5	-4	-6
D	-4	-6	-5	0	-1000	1
E	-5	-6	-2	-1000	0	-1
F	5	-1000	3	-2	-2	0

$$PG_4(I,I) = \max \left\{ \max_{\substack{J \\ J \neq I}} \left[PG_2(I,J) + PG_2(J,I) \right], PG_2(I,I) \right\}$$

If the cycle is profitable (PG > 0), it is checked for repeated nodes. If any nodes other than the ends are repeated, the cycle is not allowed. If no valid profitable cycle exists, the remainder of the matrix is formed. The four-step potential gain is computed as follows:

$$PG_4(I,J) = \max \left\{ \max_{\substack{K \\ K \neq I \\ K \neq J}} \left[PG_2(I,K) + PG_2(K,J) \right], PG_2(I,J) \right\}$$

The PG_4 matrix for Figure 2 is shown in Table 2.

Since no profitable cycles were found in the PG_4 matrix, subroutine NEXTM is called again to form the PG_8 matrix. In forming PG_8 cycles, a valid cycle A → C → D → E → F → B → A is found with profit +4. At this point, the node pairing in the profitable cycle is reversed. The new pairs are C-D, E-F, and A-B. Subroutine SOLV recomputes the PG_2 matrix. Subroutine NEXTM is called to compute the PG_4 matrix. When the PG_8 matrix is computed, no profitable cycles are found. Since the eight-step path includes all the odd-node pairs, the present pairing has a minimum total pairing length of 9.

When the optimum odd-node pairing has been obtained, PHASE3 calls subroutine TREE to find the shortest path within the section that connects each

TABLE 2. FOUR-STEP PROFIT MATRIX

START \ STOP	A	B	C	D	E	F
A	0	-1	-1000	2	4	-1
B	0	0	3	3	5	-1000
C	-1000	1	0	4	6	-4
D	6	-4	4	0	-1000	1
E	4	-1	2	-1000	0	-1
F	5	-1000	3	3	0	0

odd-node pair. The count of times each segment is to be traversed is incremented by one.

Main program PHASE3 calls subroutine TRAVEL to generate a path from the landfill or garage through the collection region and back to the landfill. The trial-and-error search used by subroutine TRAVEL is adequate to find a path because there are usually many paths having the minimum length. For example, there are more than 140 minimum-length paths starting and ending at node A in Figure 2.

Segments that are traversed more than once are placed at the beginning of the segment arrays to minimize the number of U-turns. Since the segments are examined sequentially from first to last for addition to the path, the segments requiring more than one traversal will be traveled the first time a node bounding one of them is reached. Unless the segment is a dead-end, use of this method usually precludes U-turns.

When no segments can be added to the end of the path, subroutine TRAVEL examines the segment data for segments that must still be traversed. If no unused segments remain, a suitable travel path has been found. Otherwise, the path is retraced in reverse until an unused segment is reached. At that point, the trial-and-error search resumes.

After subroutine TRAVEL has found a path for traversing the collection region, PHASE3 calls subroutine OPTPATH to perform two types of optimization

on the path. The subroutine finds the shortest distance between the points that start and end each travel stretch. Because segments that are not in the section may be used, the travel stretch may be shortened. The second type of optimization removes twice-traveled stretches on which collection is not required. If a part of the path is traveled twice in the same direction, and if there are no one-way streets on the path between the end of the first traversal and the beginning of the second, the twice-traveled piece is removed both times and the path between the first and second traversals is reversed.

As travel paths are generated for each combination of entry and exit nodes, the total length of each new travel path is compared to that of the best found thus far. When all combinations of entry and exit nodes have been tried, PHASE3 writes the best path to file TAPE9. At the conclusion of program PHASE3, TAPE9 contains two choices of travel for servicing each section, one choice starting at the garage and one starting at the landfill.

2. DATA STORAGE

Program PHASE3 obtains data from five sources: card input and files TAPE1, TAPE2, TAPE3, and TAPE4. Files TAPE1 and TAPE2 hold the segment and node data produced by program RCINPT. Files TAPE3 and TAPE4 hold the vehicle data and section list produced by program PHASE2. One file, TAPE9, is generated by program PHASE3 and is saved on disk for use by program PHASE4.

The problem title is read from the first data card at the beginning of main program PHASE3. The node numbers of the landfill and garage are read from the second data card following statement 36. A segment number and new section number are read from each remaining data card at statement 50 of PHASE3. All of the variables read from data cards are kept in storage local to PHASE3.

All of the segment data on file TAPE1 are read by a single READ statement shortly after statement 16 of PHASE3. The count of segments, NSEG, and the segment data array, STG, are stored in blank COMMON. Variable AVMD, the map distance conversion factor, is not used by the program. The parts of the STG array with second subscripts equal to 9 and 10 are not read from TAPE1.

All of the node data are read from file TAPE2 by a single READ statement shortly after statement 18 of program PHASE3. The total house count and the total refuse quantity, NHTOT and TOTREF, are kept in local storage. The remaining variables are stored in COMMON block NODDATA. The data in this COMMON block are not changed during the remainder of the program.

All of the data on file TAPE3 are read by a single READ statement. This statement is the second executable statement in PHASE3. The segment count, NSEG, is stored in blank COMMON, but will be overwritten by a subsequent READ statement. The number of sections, NTRIP, is kept in storage local to PHASE3. Arrays SINL and SFNL are stored in COMMON block TEMSTG. The pointers in these arrays are used only in the loop through statement 16 in the main program. The arrays will be overwritten by subroutine CONNECT. The array of vehicle capacities, TC, is kept in storage local to PHASE3. These values are not changed.

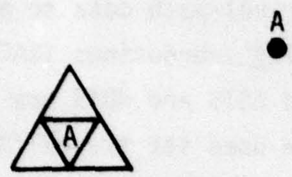
The segment numbers on file TAPE4 are ordered by section. The segment numbers are read, one at a time, by the READ statement in the loop through statement 14. The segment numbers are stored in the ISTG array, with the second subscript increasing by 1 for each segment. The section number is also stored in the ISTG array as each segment is read. Following the loop through statement 16, the segment numbers are sorted into increasing order; the section assignments are carried along. The segment numbers will be overwritten when the remaining segment data are read from file TAPE1.

Data are written to file TAPE9 from two places in main program PHASE3. Two WRITE statements shortly before statement 86 write the path from the landfill to the garage to TAPE9. The remaining path information is written by six WRITE statements immediately preceding statement 500. The statements write header cards and path descriptions for the three parts of the travel path: travel from the garage or landfill to the collection region, travel and collection within the collection region, and travel from the collection region to the landfill. These six WRITE statements are executed twice for each section, first for the trip starting at the garage, and second for the trip starting at the landfill, as part of the loop through statement 500. The loop through statement 800 repeats the loop through statement 500 for each section.

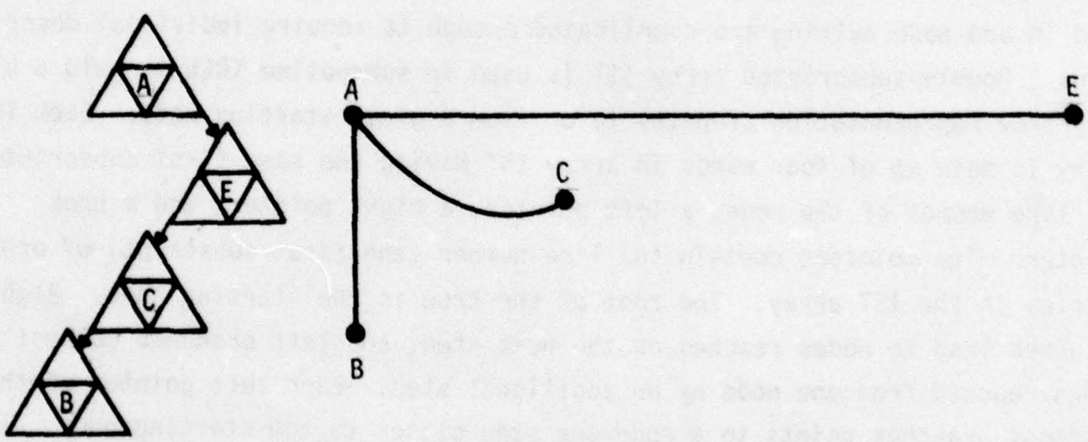
The data for the path from the landfill back to the garage come from arrays generated by the calls to subroutine TREE shortly after statement 80 of the main program. The first call to TREE generates path information for paths from the landfill node to each other node. The subsequent call to TRACE puts the path information from the garage to the landfill in arrays IPS and IPN. Subroutine FLIP reverses the order of this path. The IPN and IPS arrays are written to TAPE9. The arrays are used for other path information later in the main program. Travel-path data to and from the landfill are obtained in a similar manner, using subroutines TRACE and FLIP near statement 410 in the main program. Arrays ISTS and NDTS are used for travel to the section, and arrays ISFS and NDFS are used for travel from the section. The data are transferred to arrays ISSV and NDSV in the loops through statements 430 and 450. The data for the path in the collection region originate in subroutine TRAVEL. The path is stored in the IPN and IPS arrays. It may be changed by subroutine OPTPATH. It is transferred to arrays ISSV and NDSV in the loop through statement 440. All of the arrays used to hold path information are in COMMON block TEMSTG. These data will be overwritten by the next call to subroutine SOLV.

The tree data structure in subroutine TREE and the packed path matrices used in odd-node pairing are complicated enough to require individual descriptions. Double-subscripted array IST is used in subroutine TREE to hold a binary tree representation of paths to or from a given starting node. Each TREE entry is made up of four words in array IST having the same first subscript: the line number of the node, a left pointer, a right pointer, and a back pointer. The pointers contain the line number (the first subscript) of other entries in the IST array. The root of the tree is the starting node. Right branches lead to nodes reached on the next step, and left branches connect all nodes reached from one node by an additional step. Each left pointer of the leftmost branches points to a node one step closer to the starting node. The pointer contains the negative of this node's line number in array IST. The left and right pointers are used to scan all entries in the tree. The back pointers are used to facilitate unlinking entries as the tree is being built.

The stages of formation of a tree for the network presented in Figure 2 are shown in Figure 3. The four small triangles forming the larger triangle

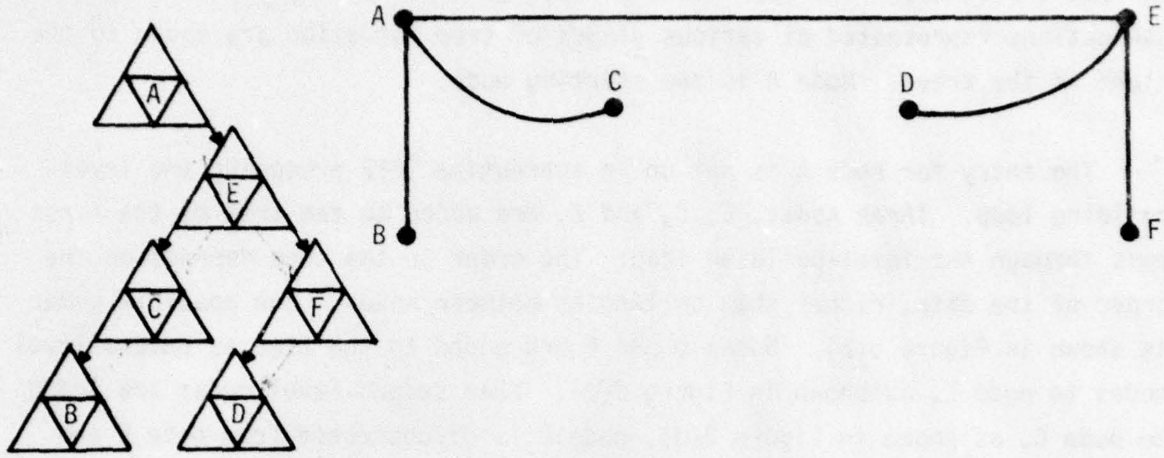


(a) Initial Setup

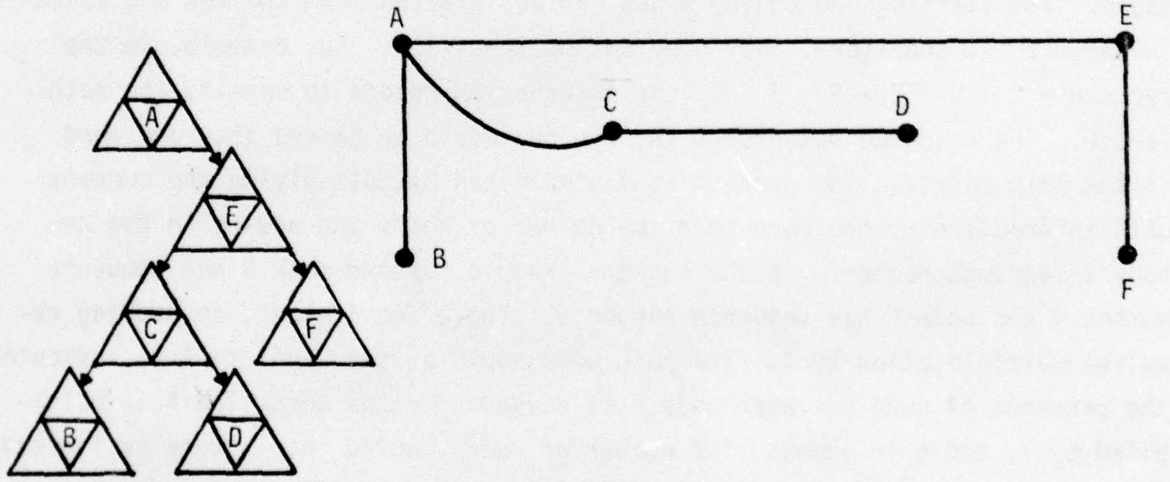


(b) First-Level Additions to Starting Node

Figure 3. Stages of Tree Formation



(c) Second-Level Additions to Node E



(d) Second-Level Addition to Node C

Figure 3. Stages of Tree Formation (Concluded)

shown in Figure 3(a) represent the four elements in each tree entry. The center triangle, corresponding to the line number of the node, contains the letter designating the node. The bottom left and right triangles represent the left and right pointers. The top triangle represents the back pointer. The network connections represented at various stages of tree formation are shown to the right of the trees. Node A is the starting node.

The entry for node A is set up in subroutine TREE preceding the level-building loop. Three nodes, B, C, and E, are added to the tree by the first pass through the level-building loop. The order in the tree depends on the order of the data, rather than on lengths between nodes. One possible order is shown in Figure 3(b). Nodes D and F are added to the tree as second-level nodes to node E, as shown in Figure 3(c). When second-level nodes are added to node C, as shown in Figure 3(d), node D is disconnected from node E and appended to node C because its distance from node A is shorter via node C than via node E. The final tree and all pointer connections are shown in Figure 4.

The packed path matrix contains only those portions of the paths which cannot be inferred. Because the paths connect pairs of odd nodes, only one node in the pair need be specified; the other is obtained from the partner table. The starting and ending nodes can be inferred from the row and column in the matrix; therefore, they need not be specified. For example, in the cycle $A \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow B \rightarrow A$, the information needed to specify the path is D,F. The sequence numbers of these nodes would be packed into one word in the path matrix. The packing is accomplished by multiplying the current path information by one more than the number of nodes and adding in the new node's sequence number. In the current example, assume node D has sequence number 4 and node F has sequence number 6. There are 6 nodes, so packing requires multiplication by 7. The path word would be set equal to 4 to indicate the presence of node D. When node F is packed into the word, the 4 is multiplied by 7, and 6 is added. The number of nodes packed into a word is limited so that 48 or fewer bits are used. A second word is available in a second matrix if needed.

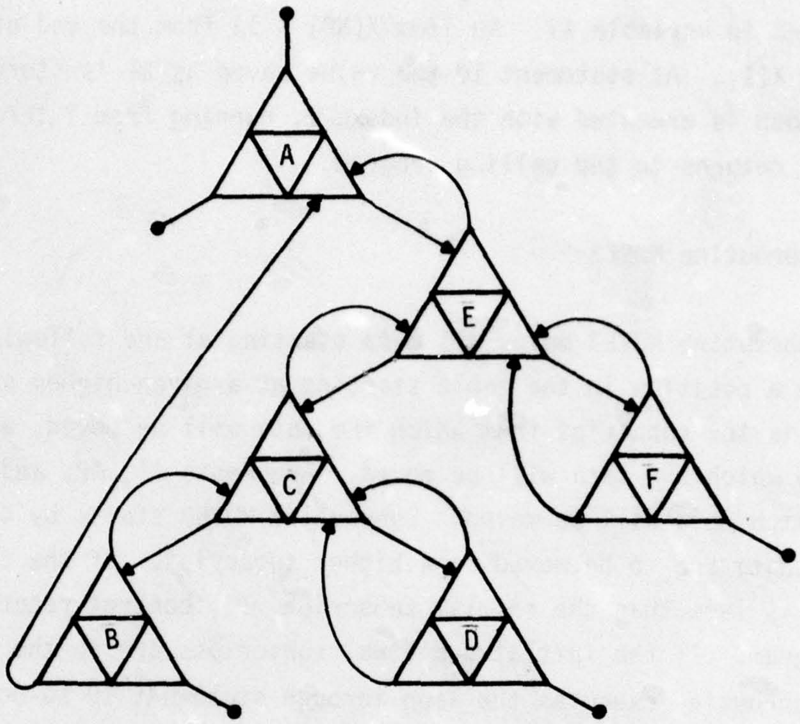


Figure 4. Final Tree for Sample Network

3. PURPOSE AND PERFORMANCE

In this section the simpler subroutines are described first so their workings will be understood when they are mentioned again in the descriptions of the more complicated subroutines. The main program is described last. Logic flowcharts are given in Appendix A. Complete program listings are provided in Appendix B. In Appendix C, the more important FORTRAN variables are described for each subroutine.

a. Subroutine FLIP

Subroutine FLIP reverses the order of an array. The first argument is the array, and the second argument is the number of items to be reversed.

Variable ND2 is set to half the number of items in the array. Variable NP1 is set to one more than the number of items in the array. The loop through statement 10 reverses the X-array. An item X(I) from the front of the

array is saved in variable XT. An item X(NP1 - I) from the end of the array is stored at X(I). At statement 10 the value saved as XT is stored at X(NP1 - I). The loop is executed with the index, I, running from 1 through ND2. Control then returns to the calling program.

b. Subroutine MOVE3

Subroutine MOVE3 moves all data starting at and following a given subscript to a position in the table starting at a given higher subscript. Argument II is the subscript from which the data will be moved, and IF is the subscript to which the data will be moved. Arguments A1, A2, and A3 are the arrays in which data will be moved. Subroutine MOVE3 starts by determining whether the data are to be moved to a higher subscript. If the final subscript, IF, is less than the initial subscript, II, control returns to the calling program. If the initial and final subscripts are in the proper relation, the subroutine executes the loop through statement 10 to move each item starting at subscript II the distance necessary to reach IF. All data between subscripts II and IF, starting with the last item, are moved. Motion starts at the higher subscript so that no data that must be moved later are overwritten. Zeros are stored at location II of the arrays. Control returns to the calling program.

c. Function IFIND

Function IFIND uses a binary search to find a given number in an array and assigns the subscript of the number as the value of IFIND. If the number is not found, the function sets the value of IFIND equal to the negative of the subscript at which the number, to be in numerical order, should be inserted. (The array is assumed to be in increasing order.) There are three arguments. Argument NUM is the number that is sought in array IARRAY. The second argument, IARRAY, is the array to be searched. Argument LEN is the length of array IARRAY.

Function IFIND begins by checking that LEN > 0. If LEN $\leq$ 0, the function assigns a value of -1 to IFIND and control returns to the calling program. This value indicates that the number sought is not in the array and would

be stored as the first entry in the array. The binary search uses variables II, IP, and IF as pointers. II is the subscript of the front of the region being searched, IP is the subscript of the item being compared to the number sought, and IF is the subscript of the last item in the region being searched. Variable II is set to 1 at statement 5. Variable IF is set to the length of the array. The pointer, IP, is the subscript midway between II and IF and is computed at statement 10.

Following statement 10, NUM is compared to IARRAY(IP). If NUM < IARRAY(IP), control transfers to statement 20, indicating that the number is in the front half of the region being searched. At statement 20 the final pointer is moved to the subscript preceding the point just searched. If NUM > IARRAY(IP), control transfers to statement 30, indicating that the number being sought follows the subscript just inspected. At statement 30 the initial pointer, II, is set to the present pointer, IP, plus 1. If the number sought is found at IARRAY(IP), control transfers to statement 50, where IFIND is set equal to the current pointer. Control returns to the calling program. Where NUM is unequal to IARRAY(IP), the initial or final pointer is moved and control resumes at statement 40. At statement 40 the final pointer is compared to the initial pointer. If IF ≥ II, control transfers to statement 10, where the search resumes on the appropriate half of the region examined previously. If the final pointer becomes less than the initial pointer, the number sought is not in the table. In this case, control resumes following statement 40, and the value of IFIND is set to the negative of the current pointer. If the number at the current pointer is less than the number being sought, IFIND is set to -(IP + 1) so the number can be inserted in the appropriate place. Control then returns to the calling program.

d. Subroutine SHLSRT3

Subroutine SHLSRT3 sorts one array and carries two other arrays along during the sorting. The subroutine uses Shell's sorting algorithm.

Subroutine SHLSRT3 has five arguments. The first, X, is the array to be sorted. The second and third arguments are arrays that are paired with X and are rearranged as X is sorted. The fourth argument is the number of words to be sorted. The fifth argument is set to +1.0 if the array is to be

sorted into increasing order or to -1.0 if it is to be sorted into decreasing order. The numbers are sorted by a procedure in which pairs of numbers are compared and interchanged, if necessary, to put the smaller number closer to the beginning of the array. The separation of the numbers compared is approximately half the number of entries in the array. This spacing is halved in subsequent passes through the array. When two numbers are interchanged, the pointers are moved up so that the smaller number is compared to a number farther up in the array.

The spacing, N, is set initially to half the number of words. The number of comparisons, K, to be performed in the loop through statement 50 is the total number of words less N. The loop through statement 50 uses index I as one of the pointers. This pointer is stored in variable J; the other pointer, L, is set to I+N. The values of the array to be sorted and the arrays to be carried along are saved as XT, AT, and BT. The values of X at the locations indicated by the pointers are compared. If they are in order, control transfers to statement 40. If not, the larger value is stored closer to the end of the array. The pointers are both moved up by N. If the pointer having the smaller value is a valid subscript, control transfers to statement 20, where another comparison is performed. At statement 40 the saved values are stored in the appropriate places in the arrays. Following the loop through statement 50, if the spacing is equal to 1, the sort is complete and control returns to the calling program. Otherwise, the spacing is halved and control transfers to statement 10.

e. Subroutine ISHLSRT

Subroutine ISHLSRT sorts one array and carries along another array during the sorting. There are five arguments. The first argument, IX, is an array of integers to be sorted. The second argument, IA, is an array that is reordered as IX is sorted. The third argument, NW, is the number of words to be sorted. The fourth argument, ISGN, is set to 1 if the array is to be sorted into increasing order or to -1 if the array is to be sorted into decreasing order. The fifth array, INC, is the spacing of the subscripts on IX and IA. The items that will be sorted and reordered fall at subscripts 1, INC + 1, 2 x INC + 1, and so forth.

The internal workings of subroutine ISHLSRT are the same as those of subroutine SHLSRT3 but for three minor differences: the array to be sorted is integer instead of floating point, only one array is carried along with the sorted array, and an increment other than 1 is allowed on the subscripts. The increment is implemented by treating arrays IX and IA as double-dimensioned arrays with a first dimension equal to INC. All references to these arrays use a first subscript equal to 1 and a second subscript which is the same as that used in subroutine SHLSRT3.

f. Subroutine ADJUST

Subroutine ADJUST puts segments meeting at order-two nodes into the same section if vehicle capacity will not be exceeded. It has two arguments. The first argument, TL, is an array of vehicle loads. The second argument, TC, is an array of vehicle capacities.

The loop through statement 70 examines each of the KNODES nodes. If the count of segments bounding a node, NUMNBR, is not equal to 2, control transfers to statement 70. Otherwise, the numbers of the segments bounding the node are obtained from the NBRSEG array and are stored in variables NSG1 and NSG2. The section assignments of these segments are obtained from the ISTG array and are stored in variables NSC1 and NSC2. If the segments are in the same section, or if either is in section zero, control transfers to statement 70. Otherwise, refuse quantities R1 and R2 are computed for the segments. If both R1 and R2 are zero, control transfers to statement 70. Otherwise, R1 is compared with R2. If R2 $\leq$ R1, control transfers to statement 30. If not, and if the sum of R1 and the load of the vehicle servicing section NSC2 exceeds the vehicle capacity, control transfers to statement 50. Otherwise, at statement 10 the load for the vehicle servicing section NSC2 is incremented by R1, and the load for the vehicle servicing NSC1 is decremented by R1. The section assignment for segment NSG1 is changed to NSC2. A message is printed indicating that the program has adjusted the section assignment. Control transfers to statement 70.

Statement 30 is reached when the refuse quantity, R2, on the second segment is less than or equal to the refuse quantity on the first segment. If

the sum of R2 and the load of the vehicle servicing section NSC1 is greater than the vehicle capacity, control transfers to statement 60. Otherwise, at statement 40, R2 is added to the load of the vehicle servicing secton NSC1 and is subtracted from the load of the vehicle servicing section NSC2. The section assignment of segment NSG2 is changed to NSC1, and a printed message indicates the change. Control transfers to statement 70. At statement 50, if the sum of R2 and the load of the vehicle servicing section NSC1 exceeds the vehicle capacity, control transfers to statement 70. If not, control transfers to statement 40. At statement 60, if the sum of R1 and the load of the vehicle servicing section NSC2 is less than or equal to the vehicle capacity, control transfers to statement 10. If not, statement 70 is reached, and the loop is repeated for the next node. After the last node has been processed, control returns to the calling program.

g. Function CUMDIS

Function CUMDIS evaluates the total length of a group of segments. It has two arguments: ISEG, an array of segment numbers, and NSG, a count of the segments.

Variable SUM is initially set to 0. The loop through statement 10 accumulates the lengths of the segments in variable SUM. If the segment number is greater than NSEG, the number of segments in the map description, the segment does not have a street number, but its segment number is obtained from the storage that normally would contain the street number. Following the loop, CUMDIS is assigned a value equal to SUM and control returns to the calling program.

h. Subroutine PRNPCS

Subroutine PRNPCS prints the segment numbers of the segments in each piece of a section. It has two arguments. The first argument, NTRIP, gives the trip or section number. The second argument, NPIECE, gives the maximum number of pieces of the section to be printed.

Variable IZERO is set to 0. The loop through statement 40 examines each of NPIECE pieces of the section. Variable ITRIP is set equal to the trip

number plus an appropriate multiple of 1000. This number is used to identify the piece of the trip. A carriage control variable, CC, is set equal to a blank. Variable NN, a count of the segment numbers on the current line of output, is set to 0. A heading giving the piece number is printed.

The loop through statement 30 scans each line of the segment table. If the piece number for the segment is not ITRIP, control transfers to statement 30. Otherwise, NN is incremented by 1. The segment number is printed. Carriage control variable CC is changed to a plus sign, causing subsequent segment numbers to be printed on the same line as the first. If fewer than 30 segments have been printed on the line, control transfers to statement 30. Otherwise, CC is set to a blank and NN is reset to 0. After the loop through statement 40 has been completed, control returns to the calling program.

i. Subroutine TRACE

Subroutine TRACE finds the segments and nodes in a path between two nodes. It has seven arguments. The first two arguments, NSTART and NSTOP, are the node numbers of the starting and stopping points in the path. The third argument, LPREVN, is an array containing the line number of the next node in the path to node NSTOP. The fourth argument, NPREVS, is an array containing the number of the next segment in the path to node NSTOP. The fifth argument, array IPS, will be filled with the segment numbers in the path by subroutine TRACE. The sixth argument, array IPN, will be filled with the node numbers in the path. The seventh argument, NSG, will be set to a count of the segments in the path from node NSTART to node NSTOP.

Function IFIND finds the line number of node NSTART in array NODNUM. The line number is stored in variable L. The starting node number is stored in IPN(1). The line number of the terminal node, NSTOP, is found by function IFIND and is stored in LEND. The segment count, NSG, is set to 0.

At statement 10 the segment count is incremented by 1. The segment number of the next segment in the path is obtained from the NPREVS array and is stored in the IPS array. The line number of the next node in the path is obtained from the LPREVN array and is stored in variable L. The number of

this node is stored in the IPN array. If the line number of this node is negative or is greater than the count of nodes, or if the count of segments in the path exceeds 200, control transfers to an error message at statement 20. Otherwise, if the line number, L, is not equal to the line number of node NSTOP, and if L is nonzero, control returns to statement 10. Otherwise, control returns to the calling program. At statement 20, an error message is printed, giving the information about the path found thus far between nodes NSTART and NSTOP. System subroutine DUMP is called to give a core dump and terminate the job.

j. Subroutine MOVODD

Subroutine MOVODD moves odd-order node pointers to the front of an array and returns a count of the odd nodes. It has four arguments. The first argument, KN, is a count of all the nodes. The second argument, NODORD, is an array of node orders. The third argument, NODPTR, is an array of node pointers. The fourth argument, NODD, is a count of odd nodes and is returned by subroutine MOVODD.

Variable JJ is set to 0. Variable KK is set to one more than the number of nodes. These variables will be pointers at opposite ends of the NODORD array. At statement 10 the front pointer, JJ, is incremented by 1. If JJ is greater than or equal to the final pointer, KK, control transfers to statement 30. Otherwise, NODORD(JJ) is examined. If it is odd, control transfers to statement 10. Otherwise, at statement 20, variable KK is decremented by 1. If JJ $\geq$ KK, control transfers to statement 30. If NODORD(KK) is even, control returns to statement 20. If not, the orders and pointers to the odd and even nodes are interchanged so that the odd-node data are at the front end of the arrays. Control returns to statement 10. Statement 30 is reached when pointers JJ and KK are equal. At statement 30 the count of odd nodes is set equal to JJ - 1. Control returns to the calling program.

k. Subroutine TREE

Subroutine TREE builds four arrays for each node that can be reached from a given starting node. The arrays give the line number of the previous node, the number of the previous segment, and the travel time and distance between the starting node and the node in question. The subroutine has nine

arguments. The first argument, NORG, is the number of the starting node. The second argument, DIST, is an array of distances from the starting node to each other node. The third argument, TIME, is an array of travel times from the starting node to each other node. The fourth argument, W, indicates whether time or distance or a linear combination of the two is minimized as paths from the starting node are built. The fifth argument, NPREVN, is an array that gives the line number of the previous node in the path for each node. The sixth argument, NPREVS, is an array that gives the number of the previous segment in the path for each node. The seventh argument, IDIR, is set to 1 if travel is from the starting node or to -1 if travel is toward the starting node. The eighth argument, MAXLVL, is the maximum number of levels or steps allowed in the path from the starting node. The ninth argument, INSECT, causes the path to be restricted to segments in section INSECT. If INSECT is 0, all segments may be used.

The items in arrays DIST, TIME, NPREVN, and NPREVS correspond to the nodes in array NODNUM. A double-dimensioned array, IST, and two single-dimensioned arrays, NIU and NIULOC, are used to build paths from the starting node. The IST array is used to construct a binary tree that corresponds to the paths from the starting node. The starting node is the root of the tree. All nodes that can be reached in one step (segment) from the starting node are added to the tree. The distances to these nodes are recorded in the NIU array, indicating that these nodes are in use. A large negative number is stored as the distance to the starting node. Nodes that are not in use have a 0 in the corresponding locations of the NIU array. All nodes that can be reached in one more step from the starting node are now added to the tree. If any node just found is already in use, and if its distance from the starting point is greater the second time it is found than the first time, it is skipped. If the earlier occurrence of the node has the greater distance, the node is deleted from the tree and all nodes reached from it are removed from the in-use category. When a pass is made in which no nodes are added and no distances are diminished, the procedure is finished. (The procedure can be made to terminate after a fixed number of steps by using the desired number as MAXLVL in the calling sequence.)

In the tree representation, the root is the starting node. A right branch leads to a node reached on the next step. The left branches connect

all nodes reached on the same step. Three pointers are associated with each element in the tree: a left pointer, a right pointer, and a back pointer. The back pointer points to the root of each subtree and is used to facilitate unlinking elements in the tree. The right pointer points to the element to the right in the tree; this pointer has a value of 0 if there is no right branch. The left pointers point to left branches. When there is no left branch, the left pointer points to the node one step closer to the starting node. NIULOC(I) gives the position in the IST array of a node with number NODNUM(I).

Following the declarative statements, six functions are defined. Function INS gives the section number of a segment, Functions NU1 and NU2 give the starting and ending nodes of a segment. Function LEN gives a segment's length. Function TIM gives the time required to travel from one end of a segment to the other. Function NWAY gives the number of ways of travel permitted on a segment.

Subroutine TREE begins execution by computing CW, the coefficient of time in the linear combination of time and distance minimized as paths are generated from the starting node. The line number of the starting node is found by function IFIND and is stored in variable I. If I > 0, control transfers to statement 570. Otherwise, an error message is printed and subroutine EXIT is called to terminate the program.

Statements 570 through 600 clear arrays IST, NIU, and NIULOC. At statement 600 the fourth column of array IST is set to the line number of the next row. These numbers are used as links for unused storage in the IST array. They will be overwritten by back pointers later in the subroutine. Variable NEXT, obtained from IST(1,4), indicates the next available line of the IST array. The line number of the starting node is stored in IST(1,1). The left, right, and back pointers for this line are set to 0. Variable CUM, the weighted function of time and distance, is set to 0. The starting node is designated in-use by the storage of -1000 in the location of array NIU corresponding to the starting node. Variable NMRT is set to 1, indicating that the current activity is at the first level of the tree. The loop through statement 940 will build up to MAXLVL levels of the tree. Variable NEWS, a new node indicator, is set to 0. Variable K, a pointer to the current line of the IST array, is set to 1.

At statement 620, variable NEXTK is set equal to the right pointer of the current tree entry. If NEXTK is 0, control transfers to statement 640. Otherwise, the level counter, NMRT, is incremented by 1. K is set equal to NEXTK, and control transfers back to statement 620.

Statement 640 is reached when an element that has no right branch has been found. At statement 640, if the level number is equal to the loop index, J, indicating that additional nodes should be added to the tree, control transfers to statement 680. Otherwise, at statement 660 NEXTK is set equal to the left pointer of the current tree entry. K is set equal to the absolute value of NEXTK. If NEXTK is negative, indicating the absence of a left branch, control transfers to statement 670. If NEXTK is 0, indicating that the NEXTK pointer is back to the root of the entire tree, control transfers to statement 930. If NEXTK is positive, indicating the presence of a left branch, control transfers to statement 620.

At statement 670, the level counter, NMRT, is decremented by 1. Control transfers to statement 660. At statement 680, variable IFIRST is set to 1 to indicate that the next node added to the tree will be a right branch rather than a left branch. The line number, N, of the node being built upon is obtained from IST(K,1). Variable LAST is set equal to K, the current node's line in the tree.

Variable KSEG is set equal to the number of neighboring segments for the node. The loop through statement 900 will examine these neighboring segments and add to the tree nodes that are one step farther from the base node. The number of a neighboring segment is retrieved and stored in variable KK. If a section restriction on segments is present, and if the segment is not in the required section, control transfers to statement 900. Otherwise, the number of the ending node of the segment is stored in variable L. If the starting node of the segment is the node already in the tree, and if travel from the starting to the ending node is permissible, control transfers to statement 700. Otherwise, if the ending node is not in the tree, or if travel from the starting to ending node is not allowed, control transfers to statement 900. Otherwise, the number of the starting node is stored in variable L.

At statement 700 the line number of node L is obtained by function IFIND and is stored in variable L. The cumulative value of the linear combination of time and distance to the node is stored in variable CUM. The location in the NIU array corresponding to the node is examined for the status of the node. If NIU(L) is negative, control transfers to statement 900. If it is 0, the node is not in use, and control transfers to statement 840. If it is positive, the node is already in use, and control transfers to statement 720.

At statement 720, CUM is compared to the value stored in array NIU. If the current value (CUM) is greater than or equal to the previous value, control transfers to statement 900. If not, the previous occurrence of the node is disconnected from the tree. The location in the tree of the previous occurrence of the node is obtained from array NIULOC and is stored in variable LB. The back and left pointers are obtained from the tree and are stored in variables LA and LC. If the node to be disconnected was not the last node added to the tree, control transfers to statement 730. Otherwise, variable LAST is set equal to the back pointer. If the node is the first node of its level added to the node of the previous level, variable IFIRST is set to 1. At statement 730 this test is repeated (although different FORTRAN variables are used); if the test is true, control transfers to statement 740. Otherwise, the left pointer of the node indicated by the back pointer is replaced by the left pointer of the current node. Control transfers to statement 760. At statement 740 the right pointer of the node indicated by the back pointer is replaced by either 0 or the current node's left pointer, whichever is larger. At statement 760, if the left pointer of the current node is positive, the back pointer of the node indicated by the current node's left pointer is replaced by the current node's back pointer. The left pointer of the current node is set to 0. Variable LC is set to the current node's line in the tree.

The statements from 780 through 840 remove the portion of the tree built on the node just disconnected. At statement 780, variable LB is set equal to variable LC. At statement 800, LC is set equal to the right pointer of the node at line LB. If the right pointer is positive, control transfers to statement 780. If not, at statement 820 the line number of the node in array NODNUM is obtained from the tree and stored in variable NA. The node's entries in arrays NIU and NIULOC are set to 0. The back pointer is set equal

to the line number of the next free storage in the tree. Variable NEXT is set equal to the line number of the node currently being disconnected. The left pointer is stored in variable LC, and LB is set equal to the absolute value of LC. If LC is negative, control transfers to statement 820. Control transfers to statement 840 if LC is zero, or to statement 800 if it is positive.

A new node is added to the tree by the processing completed between statements 840 and 900. The new node indicator, NEWS, is set to 1. The node's line number in the NODNUM array, L, is stored in the next available line of the tree. The current line's left pointer is set equal to the left pointer of the previous line. The line number of the previous node in the path is obtained from the tree and stored in array NPREVN. The segment number, KK, is stored in the NPREVS array. The right pointer is set to 0. The location of the next free storage in the tree is taken from the current entry's back pointer and is stored in variable NEWNXT. The back pointer is set equal to the line number of the previous entry in the tree. The distance and time to the current node are computed. The NIU array entry corresponding to the current node is set equal to CUM. The NIULOC array entry corresponding to the current node is set equal to NEXT, the line number of the node in the tree. If IFIRST is equal to 1, control transfers to statement 860. Otherwise, the previous entry's left pointer is set equal to the current node's line number. Control transfers to statement 880.

At statement 860 the previous entry's right pointer is set equal to the current node's line number. The current node's left pointer is set equal to the negative of the previous entry's line number. The line number of the previous node, NPREVN(L), is obtained from the previous entry in the tree. IFIRST is set equal to 0.

At statement 880, variable LAST is set equal to the current line number in the tree. Variable NEXT is set equal to NEWNXT, the line number of the next free storage in the tree. Statement 900 marks the end of the loop that scans segments connected to each node in the previous level of the tree.

A search is begun for another node in the tree to which additional nodes can be added. If any nodes have been added at the current level, K is

set equal to the negative of the left pointer of the last entry. At statement 920, K is replaced by the left pointer of the tree entry at line K. If K is positive, control transfers to statement 620. Otherwise, the level counter, NMRT, is decremented by 1. The sign of K is changed. If K is positive, control transfers to statement 920. Otherwise, NMRT is incremented by 1. If no new nodes have been added to the tree, control transfers out of the loop on statement 940 to statement 960. Statement 940 marks the end of the loop that extends the level of the tree. Following statement 960, control returns to the calling program.

1. Subroutine CON2ST

Subroutine CON2ST connects pieces of a section by two-segment paths where possible. The shortest two-segment path between pieces is used. There are two arguments. The first, NTRIP, gives the section or trip number. The second, NCON, is the count of remaining pieces requiring more than a two-step path to connect them. NCON is computed and returned by subroutine CON2ST. ISTG(10,i) contains the section and piece information for segment i. The section number identifies one piece of the section, and multiples of 1000 are added to the section number to indicate the remaining pieces.

NS, a count of two-step connections, is set to 0. The loop on statement 10 sets to 0 the arrays used for piece numbers, segment numbers, and connection lengths. The loop through statement 140 examines each node as a possible midpoint for a two-step path between pieces. The node number is stored in variable NODE. The count of segments connected to this node is stored in LIM. If only one segment is connected to the node, control transfers to statement 140. Variable LIMM1 is set equal to one less than the number of neighboring segments. The packed, neighboring-segment numbers are stored in NBRS1.

The loop through statement 130 examines all but the last neighboring segment as possible first segments of the two-segment path. The last segment will be considered only as a possible second step in a later loop. A segment number is retrieved and stored in variable ISEG. If the segment is in section NTRIP, control transfers to statement 140. Otherwise, the line number, L, of the node at the other end of the segment is found. The count of segments

bounding the second node is stored in variable LIM2. The packed, bounding-segment numbers are stored in NBRS2.

The loop through statement 20 examines each of the segments connected to the second node. If a segment is found in section NTRIP, control transfers to statement 30. Otherwise, control transfers to statement 130.

The loop through statement 120 examines the remaining segments bounding the first node. The segment number is retrieved and stored in variable JSEG. If the segment is in the current section, control transfers to statement 140. Otherwise, the line number of the other node is found and is stored in variable L. The count of segments bounding this second node is stored in LIM2. The packed segment numbers are stored in NBRS2.

The loop through statement 40 examines each segment bounding the second node. If any of the segments are in the current section, control transfers to statement 50. Otherwise, control transfers to statement 120.

At statement 50, the piece numbers connected by the two-step path are compared. If the piece numbers are equal, control transfers to statement 120. If JSECT is larger than KSECT, control transfers to statement 60. Otherwise, control transfers to statement 70. At statement 60 the values of JSECT and KSECT are interchanged so that the smaller piece number will be in JSECT. At statement 70, if either of the segments is a two-way segment, control transfers to statement 80, where the length of the two-segment path is stored in variable DIST. Otherwise, if the two one-way segments point toward or away from one other, control transfers to statement 120.

If no two-step paths have yet been found, control transfers to statement 100. Otherwise, the loop on statement 90 examines arrays NP1 and NP2 to see whether a path connecting pieces JSECT and KSECT has already been found. If a shorter connection has been saved, control transfers to statement 120. If the new connection is shorter, control transfers to statement 110. If no path connecting the pieces has been saved as yet, the loop is exited.

At statement 100, the number of entries saved is incremented by 1. The five FORTRAN statements starting at statement 110 save the piece numbers, the segment numbers, and the length of the two-step path. Statement 120 marks the end of the loop that seeks the second segment in the path. Statement 130 marks the end of the loop that seeks the first segment of the path. Statement 140 marks the end of the loop on the node connecting the two segments.

The remaining statements in the subroutine replace the higher piece number by the lower for all segments in pieces that have been connected by two-step paths. A count of the pieces, NCON, is initially set to 0. The loop through statement 170 examines each of the connections. The smaller piece number from the NP1 array is stored in variable NNEW, and the larger piece number from array NP2 is stored in variable NOLD. The smaller piece number is stored in the ISTG array for each of the segments. If the two piece numbers are equal, control transfers to statement 170. (The numbers cannot be equal on the first pass through the loop, but may be equal after the first pass.) Otherwise, IFOUND is set to 0.

The loop through statement 150 examines all of the segments in the map description and replaces the larger piece number by the smaller wherever it occurs. If any numbers are replaced, IFOUND is set to 1. After statement 150, the number of pieces, NCON, is incremented by IFOUND. If the last connection is being processed, control transfers to statement 170. Otherwise, IP1 is set equal to the line number of the next connection.

The loop through statement 160 examines the remaining connections and replaces occurrences of piece number NOLD by piece number NNEW. The numbers are rearranged, if necessary, to make the number in the NP1 array smaller than the number in the NP2 array. Following the loop through statement 170, control returns to the calling program.

m. Subroutine FNDPTH

Subroutine FNDPTH finds up to five shortest paths connecting pieces of a given trip to a given node in another piece of the trip. FNDPTH has ten arguments. The first argument, NTRIP, is the trip or section number. The

second argument, NORG, is the number of the node at which the paths must end. The third argument, IP, is the number of the piece containing node NORG. The fourth argument, NDSV, is the number of paths saved. The fifth argument, LNF, is the line number of the node that starts the path. The sixth argument, NPF, is the number of the piece in which the starting node lies. The seventh argument, LNT, and the eighth argument, NPT, are arrays containing the line number of the terminal node and the number of the piece in which it lies. The ninth argument, TIM, is an array of travel times for the paths. The tenth argument, IPATH, is a double-dimensioned array that contains the numbers of the segments in up to five paths to node NORG.

The maximum number of segments allowed in a path, MAXSTG, is evaluated by using the LOCF function on two words in the IPATH array whose second subscripts differ by one. The loop through statement 10 seeks the line number of node NORG and sets to 0 arrays DIST, TIME, LPREVN, and NPREVS. The loop through statement 40 sets the piece and line number arrays, NPT and LNT, to the values appropriate for node NORG. The embedded loop on statement 30 clears the IPATH array.

Subroutine TREE is called to build a tree, using travel time between node NORG and each other node as the variable to be minimized. The direction of travel is away from node NORG.

NDSV, the number of nodes found in other pieces, is set to 0. The loop through statement 90 will examine the piece assignment of each segment. Variable JP is set to the piece number of the segment. If the segment is not in the current section, NTRIP, or if it is in piece IP, control transfers to statement 90. Variable L is set equal to the line number of the end-point node closer to node NORG. Variable T is set equal to the travel time to this node.

Variable JJ, a pointer to the line at which nodes will be saved, is initially set to 1. If no nodes have been saved as yet, control transfers to statement 60. Otherwise, the loop through statement 50 is executed. This loop examines the piece number of each saved node. If a node is found in piece JP, the travel time to that node is compared with the travel time to the current

node. If the node already saved is reached in a shorter time than is the current node, control transfers to statement 90. Otherwise, control transfers to statement 70. When no nodes in the table are in piece JP, control continues following the loop through statement 50. Variable JJ is set to one more than the number of nodes already saved. If the maximum permissible number of nodes has not been saved, control transfers to statement 60. If the maximum has been reached, and if the travel time to the current node is greater than or equal to the largest travel time already saved, control transfers to statement 90. Otherwise, JJ is set equal to the line number of the largest time in the table. Control transfers to statement 70. At statement 60, the number of saved nodes is incremented by 1. At statement 70, the piece number of the current segment is saved. The line number of the current node and the travel time to that node are saved.

The loop through statement 80 examines the travel times to each of the saved nodes. The largest travel time is saved in variable TMAX, and the line number of this entry is saved in variable LTMAX. Statement 90 marks the end of the loop that searches for segments in pieces of the current trip.

The loop through statement 120 retrieves the paths from each saved node to node NORG. Variable NP, the number of segments in the path, is initially set to 0. Variable L is set equal to the line number in the NODNUM array of the saved node. At statement 100, NP is incremented by 1. The segment number of the next segment in the path is saved, both in the IPATH array and as variable J. The line number of the next node in the path is stored in variable L. If the segment is in piece IP [which has been saved in NPT(I)], or if the next node is node NORG, control transfers to statement 120. If not, and if the number of segments in the path is fewer than the maximum allowed, control returns to statement 100. Otherwise, an error message is printed indicating that the maximum number of steps has been exceeded. The first 15 segments in the path are also printed. System subroutine EXIT is called to terminate the program. Statement 120 marks the end of the loop that obtains paths from the saved nodes. Control returns to the calling program.

n. Subroutine CONNST

Subroutine CONNST connects pieces of section NTRIP that could not be connected by one- or two-step paths. It has two arguments: the trip number, NTRIP, and the number of pieces, NPIECE, remaining at the completion of subroutine CONNST.

Variable ITER, the count of iterations through subroutine CONNST, is set to 0. The loop through statement 10 seeks a segment that is in section NTRIP. When a segment is found, the starting node number is saved in variable NORG and control transfers to statement 20. At statement 20, ITER is incremented by 1.

Subroutine FNDPTH is called to find up to five nodes in other pieces of section NTRIP. The number of nodes found is stored in variable LIM. Variable NDC is set to 1.

The loop through statement 40 uses each of the nodes found by the first call to FNDPTH as starting points for additional calls to FNDPTH. The paths found in these subsequent calls are saved in array IPATH. The total number of paths is saved in variable NDC.

The loop through statement 60 searches the array of path times. The number of the path with the smallest time is saved in variable LTM. If no paths have been found between pieces of section NTRIP, an error message is printed and control returns to the calling program. Otherwise, at statement 80 the time for the shortest path is set to a very large number. The piece numbers connected by the path are stored in variables NPS and NPL. The number of pieces, NPIECE, is decremented by 1. The loop through statement 90 replaces each occurrence of the larger piece number in arrays NPT and NPF by the smaller piece number. The loop on statement 100 replaces the larger piece number by the smaller piece number in the segment data array, ISTG. The loop through statement 110 changes the section assignments of segments on the path between the two pieces to that of the smaller numbered piece.

The loop through statement 130 examines the piece numbers in the NPF and NPT arrays. If any numbers remain that are different from the smaller piece number, control transfers to statement 50. Otherwise, if fewer than 10 iterations have been performed and if more than one piece remains in the section, control transfers to statement 20. Otherwise, if 10 or more iterations have been performed, an error message is printed. Control returns to the calling program.

o. Subroutine CONNECT

Subroutine CONNECT connects pieces of a section. It has two arguments. The first, NTRIP, is the section number. The second, NPIECE, is the number of separate pieces remaining after the subroutine has connected all the pieces it can.

Variable II and the piece count, NPIECE, are each set equal to 1. The section number, NTRIP, is stored in variable ITRIP. The loop on statement 10 clears arrays used by subroutine TREE for path information. At statement 20 the loop through statement 30 is started. Variable II is set equal to the current value of the loop index, I. If the Ith segment is in section ITRIP, control transfers to statement 40. If the loop is completed and no segments in section ITRIP are found, program execution terminates with a STOP 30 statement.

At statement 40 the number of the starting node of the segment is stored in variable NORG. Subroutine TREE is called twice, first to build paths using only segments in section ITRIP starting at node NORG, and again to find paths to node NORG using only segments in section ITRIP. The paths are built to minimize the distance to or from node NORG.

Variable DJ is initially set to 0. Variable II is incremented by 1. If II is greater than the number of segments, control transfers to statement 80. Otherwise, a loop is begun through statement 70 to mark segments in other pieces of the current section. If the segment designated by the loop index is not in the current section, control transfers to statement 70. Otherwise, the line number of the starting node, LINE, is found. If a path exists between node NORG and the current node, control transfers to statement 70. Otherwise,

the section number of the segment is incremented by 1000. Variable DJ is set to 1, indicating that an unconnected node has been found. If DJ is 0 when the loop through statement 70 is completed, control transfers to statement 80. Otherwise, NPIECE is incremented by 1, ITRIP is increased by 1000, and control returns to statement 20.

Following statement 80, if NPIECE equals 1, control returns to the calling program. If not, the loop through statement 130 is executed. The loop examines each segment to determine whether it connects two pieces of section NTRIP. The section number is retrieved and stored in variable JSECT. If the segment is in section NTRIP, control transfers to statement 130. Otherwise, the starting and ending node numbers are saved in array NNN.

The loop through statement 110 examines each of the nodes. The count of segments connected to the node is obtained from the NUMNBR array and is stored in variable LIM. If LIM is less than or equal to 1, control transfers to statement 130. Otherwise, the loop through statement 100 is executed. This loop examines each neighboring segment. If the segment is the same as that indicated by the index of the loop through statement 130, control transfers to statement 100. Otherwise, the section number of the segment is stored in array ISECT. If the segment is in section NTRIP, control transfers to statement 110. Otherwise, control passes to the end of the loop. Following the loop through statement 100, control transfers to statement 130. Following statement 110, if the two end-point nodes are in the same section, control transfers to statement 130. Otherwise, the segment is assigned a section number formed by concatenating the piece numbers of the end-point nodes. The smaller number precedes the larger. Statement 130 marks the end of the loop that searches for single-segment connections between pieces.

After the single-segment connections have been marked, the loop through statement 160 assigns the lower piece number to all segments in the connected pieces. The connecting segment is also assigned to the piece.

The loop through statement 170 counts the number of separate pieces. The result is saved as NPIECE. If NPIECE is less than or equal to 1, control returns to the calling program. Otherwise, subroutine CON2ST is called to

make two-step connections between separate pieces of section NTRIP. Only the shortest connection between pieces is made. If the section has been merged into one piece, control returns to the calling program. Otherwise, execution continues within subroutine CONNECT.

The loop on statement 210 clears arrays NNSN and NNTN. The loop through statement 240 examines each segment. The section number is stored in variable K. If the section is not equal to NTRIP, control transfers to statement 240. Otherwise, the loop through statement 230 is executed. This loop saves the piece number and accumulates the number of houses in the piece. Following statement 240, variables NMAX and NCON are set to 0. The loop through statement 270 examines the number of houses in each piece. If the number of houses is greater than 0, control transfers to statement 260. If not, the piece number is stored in variable K. Variable NCON is incremented by 1. The loop on statement 250 removes from section NTRIP all segments in a piece that contains no houses. Control transfers to statement 270. Statement 260 saves as variable NMAX the largest piece number encountered in the loop through statement 270.

The loop through statement 280 locates the line in the NNFN array having the lowest piece number. This number should be the section number, NTRIP. If so, control transfers to statement 290. If not, program execution terminates at a STOP270 statement.

At statement 290 the house count, NNTN, is checked. If NNTN is positive, control transfers to statement 310. Otherwise, the loop through statement 300 replaces the piece numbers of segments in the highest numbered piece with the section number.

At statement 310, the number of deleted pieces, NCON, is subtracted from the total piece count. If only one piece remains, control returns to the calling program. Otherwise, subroutine CONNST is called to connect all of the remaining pieces. Control then returns to the calling program.

p. Subroutine DISCON

Subroutine DISCON removes from a section all segments that form dead-end streets on which no collection is required. The subroutine has four arguments. The first, NTRIP, gives the section number. The second, KN, gives the number of nodes in the section. The third, NODPTR, is an array of pointers to the node numbers. The fourth, NODORD, is an array of node orders.

Variable IDEL, a count of deleted nodes, is set to 0. At statement 10 the repeat indicator, IRP, is set to 0. The loop through statement 60 examines each of the nodes in the section. If the node order is not equal to 1, control transfers to statement 60. Otherwise, variable LINE is set equal to the line number of the node. The number of segments connected to the node is obtained from array NUMNBR and is stored in variable LIM. A word containing the packed segment numbers is obtained from array NBRSEG and is stored in variable NBRS.

The loop through statement 20 examines the section assignment of each neighboring segment. When a segment is found in the current section, NTRIP, control transfers to statement 30. If no segments are found in the current section, an error exists in the program. An error message is printed, and the program terminates with a call to DUMP. Otherwise, at statement 30, if the segment requires collection, control transfers to statement 60. If collection is not required, the segment is assigned to section zero. The order of the node connected to the segment is set to 0. The count of deleted nodes, IDEL, is incremented by 1. Variable NODE is set equal to the number of the node at the other end of the segment. The loop through statement 40 searches for this node. If the node is found, control transfers to statement 50. If not, an error exists in the program. An error message is printed, and program execution terminates with a call to DUMP.

At statement 50, the order of the node is decreased by 1. If this node has already been examined in the loop through statement 60, and if it now has an order equal to 1, the repeat indicator, IRP, is set to 1. Statement 60 marks the end of the loop that examines the nodes in the section. If the repeat indicator is equal to 1, control transfers back to statement 10.

When no other order-one nodes remain, the loop through statement 70 closes up the empty entries in the NODPTR and NODORD arrays. The node count, KN, is decreased by IDEL. Control returns to the calling program.

q. Subroutine EPXP

Subroutine EPXP selects nodes that are suitable points for entry into a section from the landfill or garage and nodes that are suitable points from which to leave the section for the landfill. The subroutine has four arguments. The first, KN, gives the number of nodes in the section. The second, NODPTR, is an array of pointers to the node numbers. The third, NODORD, is an array of node orders. The fourth, I, is the section number.

Variable KLIM, the maximum number of entry or exit nodes to be found, is computed on the basis of the number of nodes in the section. KLIM will have a value from 5 through 10. Variable KKEEP is set to 6. Variable KLPK is set to the number of nodes or to the sum of KLIM and KKEEP, whichever is smaller.

Data are read from file TAPE7 using a BUFFER IN statement. These data give the paths from each node to the landfill and to the garage. TAPE7 is rewound. The loop on statement 240 clears the arrays used for the entry- and exit-node numbers. The loop through statement 290 finds three types of suitable nodes: entry nodes for paths coming from the landfill, entry nodes for paths coming from the garage, and exit nodes for paths going to the landfill. The loop on statement 250 moves the travel times from the garage or landfill into the DISTS array for each node in the section. Subroutine SHLSRT3 is called to sort the times into increasing order. The NODPTR and NODORD arrays are carried along during the sort.

Variable JN, the count of saved nodes, is set initially to 0. The loop through statement 280 examines the KLPK nodes closest in time to the garage or landfill. The line number of a node is stored in variable LINE. The node number is stored in variable NS. If no nodes have been saved, control transfers to statement 270. Otherwise, the double loop through statement 260 examines the path from the current node to the garage or landfill. If the path includes a node already saved as an entry or exit point, control transfers

to statement 280. Otherwise, the count of saved nodes is incremented at statement 270. The node number is stored in the NTF array. Statement 280 marks the end of the loop that saves the entry or exit nodes. Statement 290 is the end of the loop on the three types of nodes. At statement 300 a message is printed giving the entry and exit nodes for the section. Control returns to the calling program.

r. Subroutine CLOSE1

Subroutine CLOSE1 adds to a section a path back to the section from an order-one node. The subroutine has five arguments. The first, NTRIP, is the section number. The second, KN, is the count of nodes in the section. The third, KNX, is the node count after new paths have been added to the section. The fourth argument, NODPTR, is an array of pointers to the node numbers. The fifth argument, NODORD, is an array of node orders.

Variable KNX is set equal to KN. If the trip number of the current call is unequal to the trip number of the previous call, or if no path is currently saved in array IPSSV, control transfers to statement 8. Otherwise, the path in IPSSV and any additional segments artificially created for the path are set to 0. At statement 8, the number of steps in the path is set to 0. The current section number is saved in variable NTRSV.

The loop through statement 200 examines each node in the map description, seeking order-one nodes in the current section. If the node order is not equal to 1, control transfers to statement 200. Otherwise, the line number of the node is obtained from the NODPTR array. The packed segment numbers of the neighboring segments are stored in variable NBRS. The number of neighboring segments is stored in variable LIM. If LIM is greater than 1, control transfers to statement 40. Otherwise, the street segment is a dead end.

At statement 10, the number of times the segment is to be traversed is incremented by 1. The number of the node at the end of the segment opposite the order-one node is obtained and stored in variable NODE. The loop through statement 20 finds the line number of this node, JJ, in the array of nodes for this section. At statement 30 the order of the node is incremented by 1. The

order of the order-one node is set equal to 2. Control transfers to statement 200.

At statement 40, variable MAXLVL, the maximum number of steps allowed in a return path, is set to 4. The loop through statement 50 examines the neighbors of the order-one node to obtain the segment that lies in the current section. The end-point numbers and the one-way indicator for the segment are saved.

The loop through statement 130 seeks paths, first from the section to the order-one node and then from the node to the section. If travel on the path opposes the direction of travel on the segment already in the section, control transfers to statement 130. Otherwise, the data for the segment in the ISTG array are adjusted so that the direction of travel on the segment opposes the direction of travel selected for the path. The loop through statement 75 clears the arrays used by subroutine TREE to build paths to or from a node. Subroutine TREE is then called to build paths, first to and then from the order-one node. The paths are limited to MAXLVL steps.

The loop through statement 120 examines each node in the section to find paths to the order-one node. If a node from the section is not on the path, control transfers to statement 120. Otherwise, the next segment in the path is examined. If the segment is not in the section, control transfers to statement 100. Otherwise, the line number of the next node in the path, L, and the number of the next segment in the path, M, are obtained. If the segment number is positive, control returns to statement 80. Otherwise, an error exists in the subroutine and an error message is printed. Program execution terminates with a call to subroutine DUMP.

At statement 100, the time of travel from the node at line L to the order-one node is compared to the current value of TMIN. If TMIN is less than or equal to the travel time, control transfers to statement 120. Otherwise, TMIN is set equal to the travel time. The number of segments in the path is set equal to 0. The loop through statement 110 saves the path from the node back to the order-one node. The segment numbers are stored in array IPATHS, and the node numbers are stored in array IPATHN. If a segment that is already

in the section is encountered, control returns to statement 90. Otherwise, the segments in the path are counted and the count is stored in variable NPSV. If the order-one node is encountered, control transfers to statement 120. Otherwise, the loop continues until MAXLVL steps have been saved.

Statement 120 marks the end of the loop that examines the nodes in the section as starting points for the path to the order-one node. Statement 130 marks the end of the loop that examines the two types of paths: paths traveled away from the node and paths traveled toward the node. Following statement 130, if a valid path has been found, control transfers to statement 140. Otherwise, MAXLVL is doubled. If MAXLVL is less than or equal to 32, control returns to statement 70. Otherwise, program execution terminates on a STOP1401 statement.

At statement 140, the segment connecting the order-one node to the section is restored. NPSV is set equal to 1, and the segment number is saved as the return path to the section. Variable NBRS is set equal to the segment number. Control returns to statement 10.

At statement 150, the order of the order-one node is changed to 2. The loop through statement 160 finds the node in the section at which the path begins. The order of that node is incremented by 1. The loop through statement 180 transfers the segments in the path from array IPATHS to array IPSSV. In addition, the segments are assigned to the current section, and the number of times for traversal is set to 1 for each segment. The count of nodes in the section, KNX, is incremented by 1 for each node in the path. The order of each of these nodes is set to 2.

Statement 200 marks the end of the loop that searches for order-one nodes in the section. Control then returns to the calling program.

s. Subroutine GENDM

Subroutine GENDM generates a matrix of distances from each odd node in a section to each other odd node. The subroutine has four arguments. The first, ONDMTX, is a double-dimensioned array used for the distances from each

node to each other node. The second, NODD, is the number of nodes. The third, NODPTR, is an array of pointers to the nodes. The fourth, NTRIP, is the section number.

The loop through statement 30 selects each node to be the starting point of a distance tree. The node number is stored in variable NORG. The loop through statement 10 clears the arrays used by subroutine TREE. Subroutine TREE is called to build a distance tree from node NORG, using only segments in the current section. The loop on statement 20 transfers the distances from array DIST to matrix ONDMTX. When the distance matrix is complete, control returns to the calling program.

t. Subroutine SELORD

Subroutine SELORD pairs odd nodes in a manner that comes close to minimizing the total distance between pairs. SELORD has five arguments. The first, NNN, is the dimension of the distance matrix. The second, NODD, is the number of odd nodes. The third, DIST, is a matrix of distances from each node to each other node. The fourth, IPART, is an array of sequence numbers for the partners of the nodes. This array is generated by the subroutine. The fifth argument, IDIR, selects the method used to pair the odd nodes. If IDIR equals -1, nodes are paired starting with the pair having the largest near-neighbor distance. If IDIR equals +1, nodes are paired starting with those having the smallest near-neighbor distance.

The loop through statement 20 clears arrays MINJ and IPART. Array MIND is set to 1,000,000 in this loop.

The loop through statement 60 selects each odd node as a part of a minimum-distance pair. The loop through statement 40 examines the remaining odd nodes to complete the minimum-distance pairing. If the loop indices are equal, control transfers to statement 40. Otherwise, the distance between the nodes with sequence numbers I and J is retrieved and stored in variable D. If D is greater than or equal to the current minimum distance in the MIND array, control transfers to statement 40. Otherwise, the sequence number, J, is saved in the MINJ array and the distance is saved in the MIND array. When the loop

through statement 60 is complete, the sequence number of each node's nearest neighbor will be in array MINJ. Note that some nodes may be near neighbors to more than one node.

The pairing of odd nodes begins with the largest near-neighbor distance if IDIR equals -1, or with the smallest near-neighbor distance if IDIR equals +1. The loop through statement 200 produces an odd-node pair with each execution of the loop. Variable IEXTI is set to 0. Variable IEXTD is set to either 0 or 2,000,000, depending on the value of IDIR. The loop through statement 80 examines each node. If a node already has a partner, or if the distance to its nearest neighbor makes these two nodes a less desirable choice for pairing than some other pair, control transfers to statement 80. Otherwise, the sequence numbers of the node and its nearest neighbor are saved in variables IEXTI and IEXTD. If no suitable pair has been found, control transfers to statement 200. Otherwise, the sequence numbers of the nodes are stored in the IPART array, making each node the partner of the other node.

The formation of an odd-node pair may cause one of the two nodes in a minimum-distance pair to be used. In this case, a new near neighbor must be found for the remaining node. The loop through statement 160 searches for nodes to which new near neighbors must be assigned. If the node indicated by the loop index has a partner specified in the IPART array, control transfers to statement 160. Otherwise, if the neighbor of the node does not have a partner specified in the IPART array, control transfers to statement 160. Otherwise, the near-neighbor distance for the node is set temporarily to 1,000,000. The loop through statement 140 selects a near neighbor for the node from those nodes not assigned partners in the IPART array.

Statement 160 marks the end of the loop that selects new minimum-distance pairs. Statement 200 marks the end of the loop that generates odd-node pairs. Control returns to the calling program.

u. Subroutine PATH

Subroutine PATH takes the packed representation of two paths meeting at a node and produces both a packed and an unpacked version of the path from

the first to the last node. The subroutine has six arguments. The first, ISTPO, is a three-dimensional array of packed paths. The second, ISTPN, is a three-dimensional array of packed paths produced by subroutine PATH. The third argument, II, is the sequence number of the node that starts the first path to be examined by the subroutine. The fourth argument, KK, is the sequence number of the node that ends the first path and starts the second path. The fifth argument, JJ, is the sequence number of the node that ends the second path. The sixth argument, NG, is an error indicator. NG is set to -1 when the new packed path exceeds the available storage, or to 0 when no problems are found in the combined path. NG is set to 1 if a node in the first path occurs a second time in the second path.

Variable IDIM, the number of words available for each packed path, is set to 2. Variable JDIM, the maximum number of nodes used in the paths, is evaluated using the locations of the JSTP and ISTP arrays. The loop on statement 10 clears array ISTP, which is used to hold the node numbers in the unpacked path. Variable NE, the number of nodes in the path, is set to 0. Variable I is set equal to the sequence number of the starting node of the first path, and variable J is set equal to the sequence number of the node ending the first path.

The loop through statement 100 unpacks each of the two packed path words. Variable MID is set equal to one more than the number of nodes stored in the ISTP array. The nodes corresponding to variable I and the paired partner to that node are stored in consecutive locations in array ISTP. The count of nodes, NE, is incremnted by 2. Variable NW, a pointer to the appropriate packed path word, is set equal to 1. At statement 20 the packed path word is moved from matrix ISTPO to variable IWD. At statement 40, if IWD is positive, control transfers to statement 60. Otherwise, NW is incremented by 1. If NW is less than or equal to IDIM, control transfers to statement 20. Otherwise, control transfers to statement 80.

The two statements starting at statement 60 retrieve the next node, M, from the packed path. Node M and its partner are stored in the next two available locations of array ISTP. The count of nodes is incremented by 2. The remainder of the packed path is transferred from variable NIWD to variable IWD. Control transfers back to statement 40.

At statement 80, pointer J is transferred to variable I. Variable J is set equal to the sequence number of the final node of the second path. Statement 100 marks the end of the loop that examines the two packed path words.

Following statement 100, the count of nodes is incremented by 1, and JJ is stored as the last entry in the ISTP array. Variable MIDM is set equal to one less than the number of nodes in the first path, and MIDP is set equal to one more than the number of nodes in the first path.

Error indicator NG is set to 1. The double loop through statement 120 searches the second path for a node, other than the starting or stopping node, that has already occurred in the first path. If a node occurs in both paths, control returns to the calling program. Otherwise, following statement 120, the error indicator is set to 0.

The loop through statement 160 combines the two packed paths into one packed path. The loop is executed once for each word in the packed path. Pointers to the starting and stopping nodes in a packed path word are computed and stored in variables KI and KF. The packed path word in the ISTPN array is set to 0. Variable IWD is set to 0. If the final subscript is less than the initial subscript, control transfers to statement 160. Otherwise, the loop through statement 140 builds the packed path word, accumulating the result in variable IWD. At statement 160, variable IWD is transferred to the appropriate location in the ISTPN array. Following statement 160, if the number of nodes in the packed path causes the available storage for packed paths to be exceeded, variable NG is set to -1. Control returns to the calling program.

v. Subroutine NEXTM

Subroutine NEXTM doubles the maximum number of steps in a path while seeking more profitable paths from one node to another. The subroutine has six arguments. The first, MTXO, is a matrix of profits in the paths between nodes. The second, MTXN, is the new matrix of profits produced by the subroutine when the maximum number of steps in each path is allowed to double. The third argument, NODD, is the number of nodes. The fourth and fifth arguments, ISTPO and ISTPN, are matrices of packed paths for the original paths and the new paths

produced by the subroutine. The sixth argument, NRES, indicates whether or not a profitable cycle has been found.

Variable NRES, the profitable-cycle indicator, is set to 0. The number of nodes is stored in variable NEDGE. Variable MAXD is set to 0.

The loop through statement 160 selects each node in turn as the starting and stopping node of a cycle. The next four statements transfer the profit and path for the cycle from the old matrices to the new matrices.

The loop through statement 140 examines each node as the midpoint of a cycle. If the midpoint node is equal to the starting node, control transfers to statement 140. Otherwise, the profit of the path from the starting node to the center and then back to the starting node is computed and stored in variable MPR. If MPR is less than or equal to the current profit of the cycle, control transfers to statement 140. Otherwise, subroutine PATH is called to produce the packed path for the new cycle. If error indicator NG is nonzero, control transfers to statement 140. Otherwise, the new profit is stored in the profit matrix. If the new profit is less than or equal to MAXD, control transfers to statement 140. Otherwise, MAXD is set equal to the new profit. NES is set equal to the number of steps in the path. The loop on statement 120 transfers the nodes in the current profitable cycle from the ISTP array to the JSTP array. Statement 140 marks the end of the loop that selects the midpoint node of each cycle. Statement 160 marks the end of the loop on the nodes.

If MAXD is essentially zero (it is compared to a small number to take roundoff errors into account), control transfers to statement 180. Otherwise, the profitable-cycle indicator is set equal to 1 and control returns to the calling program.

Following statement 180, a double loop through statement 240 is begun. The loop seeks more profitable paths from one node to another. If the two nodes are the same, control transfers to statement 240. Otherwise, MAXP is set equal to the current profit in the path from node I to node J. The two words in the packed path are transferred from matrix ISTPO to matrix ISTPN. The profit is transferred to matrix MTXN. The loop through statement 120 selects

each node to be the midpoint of a new path from node I to node J. If the node selected is either the starting or ending point, control transfers to statement 220. Otherwise, variable MPR is set equal to the profit of the new path. If MPR is less than or equal to MAXP, control transfers to statement 220. Otherwise, subroutine PATH is called to produce the new packed path between the nodes. If the error indicator is nonzero, control transfers to statement 220. Otherwise, the new profit is stored in both matrix MTXN and in variable MAXP. Statement 220 marks the end of the loop on the midpoint nodes. Statement 240 marks the end of the double loop on the nodes. Control returns to the calling program.

w. Subroutine SOLV

Subroutine SOLV improves the odd-node pairing until the minimum total pairing distance is obtained. The subroutine has three arguments. The first, DIST, is an array of distances between nodes. The second, NODD, is the number of odd nodes. The third, IERR, is an error indicator. The subroutine examines all closed paths of a given length constructed from odd-node pairs. If the cycle produces a shorter total pairing distance when the end node of each pair is paired with the first node of the next pair, the new pairing is used. The algorithm then repeats, starting with two-step paths. If no profitable cycle is found, the algorithm doubles the number of steps allowed in the path. If no profitable cycle has been found when all odd-nodes pairs have been used in the path, the optimum odd-node pairing has been obtained.

Variable NCH, a count of iterations through subroutine SOLV, is set to 0. Variable NNPW, the number of nodes that can be packed into a word, is computed. Variable NN is set equal to 2. The loop through statement 40 doubles NN until it exceeds the number of odd nodes. When NN exceeds the number of odd nodes, control transfers to statement 60. Variable LIMK has a value of one less than the loop index. The double loop through statement 220 generates path and profit matrices for the most profitable paths, involving two steps, between any two nodes. The profit from any node to its partner is set equal to -1000.

The loop through statement 300 doubles the number of steps in the path until the maximum number of steps is obtained. If the loop index is even,

control transfers to statement 240. Otherwise, subroutine NEXTM is called to double the number of steps used in the paths. Matrices MTX1 and ISTP1 contain the old profits and paths; matrices MTX2 and ISTP2 receive the new profits and paths. If the profitable-cycle indicator is nonzero, indicating a profitable cycle, control transfers to statement 400. Otherwise, control transfers to statement 300.

At statement 240, subroutine NEXTM is called. This time, however, MTX2 and ISTP2 are the old matrices, and MTX1 and ISTP1 are the new matrices. If a profitable cycle is found, control transfers to statement 400. Otherwise, the loop continues. Following the loop, control returns to the calling program.

Following statement 400, the iteration count is incremented by 1. If fewer than 20 iterations have been performed, control transfers to statement 440. Otherwise, an error message is printed. The error indicator, IERR, is set to 1, and control returns to the calling program.

At statement 440, a loop through statement 460 is begun. This loop selects an alternative pairing of odd nodes from the nodes in the profitable cycle. Control returns to statement 120.

x. *Subroutine TRAVEL*

Subroutine TRAVEL finds a path that travels all of the segments in a section, starting and stopping at specified nodes. It has 12 arguments. The first and second, IST and ISP, are the starting and stopping node numbers. The third, NSEG, is the number of segments. The fourth and fifth arguments, NN1 and NN2, are arrays of the starting and ending nodes of the segments. The sixth argument, NSG, is an array of segment numbers. The seventh, NWAY, is an array of one-way indicators. The eighth, NTIMES, is an array that specifies the number of times each segment must be traversed. The ninth argument, NN, is the number of segments in the final path. The tenth and eleventh arguments, IPN and IPS, are arrays for the node and segment numbers in the path. These arrays are generated by the subroutine. The twelfth argument, IERR, is an error indicator.

Error indicator IERR and pointer IPT1 are set initially to 1. Variable M is set equal to KSDIM, the dimension of the arrays used in bulding the path. Pointer IPT2 is set equal to the number of segments.

At statement 10, if IPT2 is less than IPT1, control transfers to statement 40. Otherwise, if the segment indicated by IPT1 is to be traversed only once, control transfers to statement 15. Otherwise, pointer IPT1 is incremented by 1. Control transfers back to statement 10. At statement 15, if the segment indicated by IPT2 is to be traversed twice, control transfers to statement 30. Otherwise, pointer IPT2 is decremented by 1. Control transfers to statement 10.

The 15 statements starting at statement 30 interchange the data for the segments indicated by pointers IPT1 and IPT2. The interchange puts all segments that are to be traversed twice at the front of the arrays. Pointer IPT1 is incremented by 1, and pointer IPT2 is decremented by 1 following the interchange. Control transfers back to statement 10.

At statement 40, a loop on statement 55 is begun. This loop transfers the number of times the segments are to be traversed from the NTIMES array to the KTIMES array. The loop on statement 205 clears the KS array. This array counts the number of times the subroutine backs up to each step in the path during the trial-and-error search for the path.

The starting node number is stored as the first entry in the IPN array. The number of steps currently in the path, NN, is set to 1. The first entries in the KLINE and NEXTII arrays are set to 0. The KLINE array indicates the line number in the segment data from which a particular step in the path comes. The NEXTII array gives the line number in the segment data at which a search will begin for a different segment as a particular step in the path. At statement 65, II is set initially to 1.

At statement 90, a loop begins through statement 60. This loop will examine segments, starting at line II of the segment data, to find a segment that can continue the current path. If the segment indicated by the loop index cannot be used again, or if its use would create a U-turn, control transfers

to statement 60. Otherwise, if the starting node of the segment is the same as the last node currently in the path, control transfers to statement 70. If not, and if the segment is a one-way segment, or if the ending node of the segment is not the last node currently in the path, control transfers to statement 60. Otherwise, the current segment is added to the path. Its number is stored in array IPS, and its starting node is stored in array IPN. The count of steps in the path is incremented by 1. The line number of the segment is stored in the location in array KLINE corresponding to the current step in the path. The location of the NEXTII array corresponding to the current step in the path is set equal to the current loop index plus 1. The availability count for the segment is decremented by 1. Control returns to statement 65.

At statement 70 the segment is added to the path. Its number is stored in the IPS array. The node count is incremented by 1. The ending node number is stored in the IPN array. The line number of the current segment and the loop index plus 1 are stored in the appropriate locations of arrays KLINE and NEXTII. The availability count for the segment is decremented by 1. Control returns to statement 65. Statement 60 marks the end of the loop that searches the segment data for a suitable segment to be added to the path.

When control passes beyond the loop through statement 60, no suitable segment has been found for addition to the path. The line number of the last segment currently in the path is stored in variable I. If this segment is not available to be used again, control transfers to statement 75. Otherwise, the segment is added as the next step in the path, creating a U-turn. The node count is incremented by 1, and the appropriate node is appended to the IPN array. The line number of the segment is stored in the appropriate place in the KLINE array. The count of segments plus 1 is stored in the NEXTII array, indicating that no segments remain as possibilities for this step. The availability count for the segment is decremented by 1, and control transfers to statement 65.

At statement 75, if the last node in the path is not the appropriate stopping node, control transfers to statement 85. Otherwise, the loop through statement 80 examines the availability of each segment. If each segment has

been used the appropriate number of times, control transfers to statement 80. Otherwise, the line number of the segment is stored in the IPN array, starting at the end of the array. A pointer, M, is decremented by 1. Following statement 80, if M does not point to the end of the array, control transfers to statement 185. Otherwise, the error indicator, IERR, is cleared. Following statement 201, if an error condition is present, an error message is printed. Control returns to the calling program.

At statement 85, variable I is set equal to the line number of the last segment in the path. If this line number is less than or equal to 0, control transfers to statement 201. Otherwise, the availability count of the segment is incremented by 1. Variable II is set equal to the value of NEXTII corresponding to the current step in the path. The location in the KS array corresponding to the current step is incremented by 1. If the subroutine has backed up to this step in the path more than MAXKS times, control transfers to statement 201. Otherwise, the number of steps in the path is decreased by 1. If any segments remain as possible replacements at this step in the path, control transfers to statement 90. Otherwise, control transfers to statement 85.

At statement 185, pointer M is incremented by 1. It now points to the first line number of an unused segment in the IPN array. At statement 186, the line number of the last step in the path is stored in variable I. If the line number is less than or equal to 0, control transfers to statement 201. Otherwise, the availability count of the segment is incremented by 1. Variable II is set from the NEXTII array. The appropriate location in the KS array is incremented by 1. If this value exceeds MAXKS, control transfers to statement 201.

At statement 210, NN is decremented by 1. The loop through statement 83 examines the unassigned segments to determine whether one of them can be used as the next step in the path. If a suitable segment is found, control transfers to statement 84. Otherwise, control transfers to statement 186.

At statement 84, M is restored to its maximum value, KSDIM. If variable II indicates that segments remain to be examined, control transfers to statement 90. Otherwise, control transfers to statement 85.

y. Subroutine OPTPATH

Subroutine OPTPATH performs two types of optimization on the traversal path produced by subroutine TRAVEL: first, it finds the shortest path from the start to the end of each travel-only sequence of segments; and second, it removes sequences of travel-only segments that are traveled twice in the same direction. The subroutine has five arguments. The first and second, NNP and NSP, are arrays of the nodes and segments in the travel path. The third, TRTY, is an array that indicates for each segment whether both travel and collection or travel, only, will be required. This array is generated by the subroutine. The fourth argument, N, is the number of nodes in the path. The fifth argument, ISC, is the section number.

Variable NM1 is set equal to N - 1. The loop through statement 20 sets the travel type for each of the NM1 segments. The travel type is initially set to T to indicate travel only. If collection is not required on this segment, control transfers to statement 20. Otherwise, the travel type is changed to a C to indicate that collection is required. If the segment is traveled more than once, the number of times of traversal is set to 0. This prevents the segment from being marked for collection twice. Statement 20 marks the end of the loop that sets the travel type.

The loop through statement 30 seeks the first segment on which collection is required. When this segment is found, control transfers out of the loop to statement 40. At statement 40, the segments, nodes, and travel types in the path are moved forward so that the first collection segment will be the first segment in the path. The number of nodes in the path description is recomputed.

Variable IF is set initially to 1. At statement 60, a loop begins through statement 70. This loop examines each segment, seeking a segment on which only travel is required. If such a segment is found, control transfers to statement 80. Otherwise, control transfers to statement 200.

At statement 80, a loop through statement 90 is begun. This loop seeks the next segment requiring collection. If such a segment is found,

control transfers to statement 100. Otherwise, control transfers to statement 200.

At statement 100, the number of consecutive traveled segments, IS, is computed. If IS is equal to 1, control transfers to statement 60. Otherwise, variable NS is set to 0. If the traveled segments start and end at the same node, control transfers to statement 115. Otherwise, the loop through statement 110 clears the arrays used by subroutine TREE. Subroutine TREE is called to find the shortest (in time) paths starting at the node that ends the travel stretch. Subroutine TRACE is called to obtain the node numbers, segment numbers, and number of segments in the shortest path from the starting to the stopping node.

If the path neither starts nor ends with a U-turn, control transfers to statement 115. Otherwise, function CUMDIS is used to find the travel distances of the original and the new travel paths. If the new path saves less than half a mile and the old path is less than three times the length of the new path, control returns to statement 60. Otherwise, at statement 115 variable IDEL is set equal to the number of segments by which the new path is longer than the old path. The number of nodes in the trip using the new path is computed and stored in variable NEWN. If IDEL is negative, control transfers to statement 140; if it is zero, control transfers to statement 160; and if it is positive, control transfers to statement 120.

Statement 120 and the loop through statement 130 move path information in arrays NNP, NSP, and TRTY to make room for the additional segments in the new path. Control then transfers to statement 160.

Starting at statement 140, the loop through statement 150 moves the data in the travel path forward by the number of segments by which the new path is shorter than the old path.

At statement 160, if there are no steps in the new path, control transfers to statement 180. Otherwise, the loop through statement 170 moves the new travel path on top of the old travel path in arrays NNP, NSP, and TRTY. At statement 180, variable IF, the number of the last step in the path

examined thus far, is recomputed to take into account the change in the path length. Variable N is set equal to variable NEWN. Variable NM1 is recomputed. Control returns to statement 60.

At statement 200, variable DO is set to 0. Variable NM1 is set equal to N - 1, and NM3 is set equal to N - 3. The loop through statement 260 selects, in turn, each segment in the travel path except the last two. If the segment's travel type indicates collection, control transfers to statement 260. Otherwise, variable II is set equal to the loop index, variable JI is set equal to the loop index plus 1, and variable NS is set equal to the segment number.

The loop through statement 250 examines the remaining segments. If the segment number is not equal to NS, control transfers to statement 250. If the segment number equals NS, and if the segment requires collection or is traveled in the opposite direction, control transfers to statement 250. Otherwise, variable L is set equal to 1. The loop through statement 210 increments L for each identical step in the two paths starting at the two occurrences of segment NS.

At statement 220, variable IF is set equal to the number of the first step following the first of the two identical travel stretches. Variable JF is set equal to the number of the last step before the beginning of the second travel stretch. The loop through statement 230 examines the steps between IF and JF, seeking a one-way segment. If a one-way segment is found, control transfers to statement 250. Otherwise, variable DN is set equal to 0. The loop on statement 240 computes the length of the duplicated travel stretch. If the length of the stretch, DN, is less than or equal to DO, control transfers to statement 250. Otherwise, DO is set equal to DN. Pointers are saved, marking the starts of the two identical travel stretches and the starts of the segments between them.

Statement 250 marks the end of the loop seeking the second occurrence of a segment. Statement 260 marks the end of the loop selecting the first segment. If no duplicated travel stretch has been found in these loops, control transfers to statement 310.

If a duplicated travel stretch has been found, pointers are set to the first and last steps in each travel stretch. The 11 statements starting at statement 270 reverse the path between the two travel stretches. The first travel stretch is overwritten during the reversal. The loop through statement 280 moves the remainder of the travel path forward in arrays NSP, TRTY, and NNP, so that it immediately follows the portion of the path that has been reversed. The count of nodes in the entire path is recomputed. Control returns to statement 200.

At statement 300, the count of nodes is decremented by 1. Variable NM1 is recomputed. At statement 310, if the last segment in the travel path does not require collection, control returns to statement 300. Otherwise, control returns to the calling program.

z. Program PHASE3

The problem title is read from the first data card. The number of segments, number of sections, pointers to the first and last segment in each section, and vehicle capacities are read from file TAPE3. The problem title is printed, followed by the heading for the initial segment assignments. The loop through statement 16 controls the reading of segments in each section. Variables JI and JF are set equal to the sequence numbers of the initial and final segments in the section. The loop through statement 14 reads the segments in the current section. The segment number is stored in the portion of the ISTG array whose first subscript is equal to 1. At statement 14, the section number is stored in that portion of the ISTG array whose first subscript is equal to 9 (symbolically NSECT). At statement 16 the section number and the numbers of the segments in that section are printed. Subroutine ISHLSRT is called to sort the segment numbers into increasing order. The section numbers are carried along during the sort.

The segment data are read from file TAPE1 into the STG array. Note that the section assignments are not overwritten. The node data are read from file TAPE2. The loop through statement 19 examines each segment. If the number of houses on the segment is 0, the section assignment is set to 0. At statement 19, negative house counts are made positive. The loop through statement 20 accumulates the total refuse quantity for each section.

The loop through statement 36 computes the number of neighboring segments for each node. The computation examines the size of the word containing the packed neighboring-segment numbers.

The node numbers of the landfill and garage are read from the second data card. A heading is printed for section assignment changes. Subroutine ADJUST is called to put segments that meet at order-two nodes into the same section, wherever possible. The section assignment changes are printed by subroutine ADJUST. The load and capacity of each vehicle are printed.

At statement 50, the segment number and the new section assignment for the segment are read from the next data card. If an end-of-file card is encountered during the read, control transfers to statement 80. Otherwise, control resumes at statement 70. At statement 70, the old section assignment for the segment is transferred to variable NSCO. The new section assignment is stored in the ISTG array. If the new section assignment is less than or equal to 0, control transfers to statement 72. Otherwise, the refuse quantity on the segment is computed. The vehicle loads for the old and new sections are adjusted accordingly. At statement 72, the section reassignment is printed. Control returns to statement 50.

Following statement 80, the final vehicle loads and capacities are printed. Segment data, including the latest section assignments for the segments, are also printed.

Subroutine TREE is called three times. On the first call, minimum-time paths from the landfill node to the other nodes are generated, with the travel direction away from the landfill. On the second call, minimum-time paths from the garage node to each other node are generated, with the direction of travel away from the garage. On the third call, minimum-time paths from the landfill node to each other node are generated, with the direction of travel toward the landfill.

Subroutine TRACE is called to find in the data the path from the garage to the landfill, where travel direction will be away from the landfill. Subroutine FLIP is called twice; first, to reverse the order of the segments

in the path and second, to reverse the order of the nodes in the path. The distance from the landfill to the garage is evaluated by function CUMDIS and is stored in variable DIST. The path from the landfill to the garage is written to file TAPE9.

The data describing the paths to and from the landfill and garage are written to file TAPE7 by a BUFFER OUT statement. File TAPE7 is rewound.

The node numbers of the landfill and garage are printed, followed by the data describing paths to and from these nodes.

Carriage-control variable CC is set equal to the character 1. The loop through statement 180 examines each of the nodes. If the node is connected to the landfill, control transfers to statement 140. Otherwise, an error message is printed. The carriage control is changed to a blank. At statement 140, if the node is connected to the garage, control transfers to statement 180. Otherwise, an error message is printed. Following the loop through statement 180, the carriage control is examined to determine whether an error has occurred. If no disconnected nodes have been found, control transfers to statement 200. Otherwise, an error message is printed and program execution terminates with a call to subroutine EXIT.

The loop through statement 800 generates the routes for each section. The loop on statement 202 transfers the permanent section assignments to storage in the ISTG array for temporary section assignments. Subroutine CONNECT is called to connect separate pieces of the current section. In this subroutine, the temporary section assignments of segments needed to connect pieces of the current section are set equal to the current section number. The loop on statement 210 clears the node order and node pointer arrays.

Variable KN, the count of nodes in the section, is initially set to 0. The loop through statement 230 examines each segment. If the temporary section number is not equal to the current section number, control transfers to statement 230. Otherwise, the loop through statement 220 examines the segment's end-point nodes. If the line number of the node is already stored

in the NODPTR array, control transfers to statement 220. Otherwise, subroutine MOVE3 is called to make room for the node in the NODPTR and NODORD arrays. The line number of the node is stored in the NODPTR array. At statement 220, the order of the node is incremented by 1. Statement 230 marks the end of the loop that examines each segment.

Subroutine DISCON is called to remove from the section any dangling (dead-end) segments on which no refuse collection is required. During the execution of this subroutine, nodes may be deleted from the node order and pointer arrays. When control returns from the subroutine, variable KN will contain a count of the nodes in the current section.

Subroutine EPXP is called to select good entry and exit nodes for the section. The entry and exit nodes are stored by the subroutine in array NTF in COMMON block TEMSTG.

The loop through statement 240 saves the current values of the node orders and pointers in arrays NODORDS and NODPTRS. The current number of nodes in the section is saved in variable KNS.

The loop through statement 500 selects, in turn, the two possible starting points for the trip: first, the garage and second, the landfill. Variable SAVDIS is set equal to a very large number. The loop through statement 470 examines up to JMAX entry-point nodes. The entry-point node number is stored in variable ISTART. If ISTART is 0, control transfers to statement 480. Otherwise, the loop through statement 460 selects up to KMAX exit nodes from the section. The exit-node number is stored in variable ISTOP. If ISTOP is 0, control transfers to statement 470. Otherwise, the count of nodes is restored to variable KN from variable KNS.

The loop through statement 250 examines each node in the section. The node's order and pointer are restored to arrays NODORD and NODPTR. If the node is the entry node for the section, its order is incremented by 1. At statement 250, if the node is the exit node from the section, its order is incremented by 1.

The loop through statement 260 examines each segment in the map description. The availability count (number of traversals) is initially set to 0. If the temporary section assignment indicates that the segment is needed in the current section, the availability count is set equal to 1. Subroutine CLOSE1 is called to generate paths from order-one nodes back to the section. This subroutine may cause additional nodes and segments to be added temporarily to the section. The availability counts are incremented accordingly. The node count, KN, is set equal to the count, KNX, returned by subroutine CLOSE1.

Subroutine MOVODD is called to move the odd-order nodes to the front of the node number and pointer arrays. If the count of odd nodes is less than or equal to the maximum (32), control transfers to statement 270. Otherwise, an error message, including node numbers and orders, is printed to indicate that the current section contains too many odd nodes. Subroutine PRNPCS is called to print the numbers of the segments in the section. Control transfers to statement 460.

Following statement 270, subroutine GENDM is called to generate a matrix of distances between pairs of odd nodes. Subroutine SELORD is called to pair the odd nodes in a manner that approximately minimizes the total pairing distance. Error indicator IERR is set to 0. If there are more than two odd nodes, subroutine SOLV is called to generate the odd-node pairing having the minimum pairing distance. If an error has been encountered, control transfers to statement 460. Otherwise, execution continues with the loop through statement 290, which examines the odd nodes. If the sequence number of an odd node exceeds that of its partner, control transfers to statement 290. Otherwise, pointers to the nodes are stored in variables N1 and N2. The loop through statement 280 clears arrays to be used by subroutine TREE. Subroutine TREE is called to generate minimum-distance paths from one of the odd nodes to the remaining nodes in the section. The number of traversals for each segment on the shortest path between the odd nodes is incremented by 1. Statement 290 marks the end of the loop that examines the odd nodes.

The loop through statement 300 examines each segment in the map description. If the number of traversals is less than or equal to 2, control transfers to statement 300. Otherwise, if the street is not a one-way street,

the number of traversals is reduced by a multiple of 2 until it is either 1 or 2.

The loop on statement 310 clears the IPN and IPS arrays. Variable JSEG is set equal to 0. The loop through statement 380 examines each segment. If the segment is not needed in the current section, control transfers to statement 380. Otherwise, the segment number, end-point node numbers, availability count, and number of ways of travel are transferred to arrays JNSG, JNN1, JNN2, JTIM, and JNWY. Subroutine TRAVEL is called to find a traversal path from the entry-point node to the exit-point node, using the segments in these arrays. If an error is encountered, an error message is printed. If no error has been encountered, subroutine OPTPATH is called to improve the traversal path. The node numbers of the starting and ending nodes are stored in variables ISTART and ISTOP.

The data describing paths to and from the landfill and garage are read from TAPE7 by a BUFFER IN statement. If IERR indicates the presence of an error, control transfers to statement 460. Otherwise, if the current trip starts at the landfill, control transfers to statement 410. If not, subroutine TRACE is called to obtain a path from the starting node to the garage, where travel will be away from the garage. Control transfers to statement 420. At statement 410, subroutine TRACE is called to obtain the path from the starting node to the landfill, where travel will be away from the landfill. At statement 420, subroutine FLIP is called to reverse the order of the segments in the path. It is called again to reverse the order of the nodes. Subroutine TRACE is called again to obtain the path from the exit node to the landfill.

Function CUMDIS evaluates the distance for the three parts of the trip. The total distance is stored in variable DIST. If DIST is greater than or equal to SAVDIS, control transfers to statement 460. Otherwise, SAVDIS is set equal to DIST. Each of the loops through statements 430, 440, and 450 saves one of the three parts of the trip. The number of segments in each part and the length of each part are also saved.

Statement 460 marks the end of the loop that selects the exit node. Statement 470 marks the end of the loop that selects the entry node.

At statement 480, if a valid trip was found, control transfers to statement 490. Otherwise, an error message is printed and control transfers to statement 500. Starting at statement 490, the three parts of the saved trip are written to file TAPE9. This trip has the shortest total distance for all of the entry- and exit-node combinations that have been examined.

Statement 500 marks the end of the loop that selects either the garage or the landfill as the start of the trip. Statement 800 marks the end of the loop that generates routes for the sections. An end-of-file is written on TAPE9. Subroutine EXIT is called to terminate program execution.

(The reverse of this page is blank.)

SECTION IV
INPUT AND OUTPUT

1. INPUT

Input to program PHASE3 consists of card input and four disk files. Two of the disk files are generated by program RCINPT, and two are generated by program PHASE2. The five files are read at or near the beginning of main program PHASE3.

a. Card Input

The form and contents of the three types of data cards are shown in Table 3. Types 1 and 2 are required. The first contains a title that is printed on the first line of the output. The second contains the node numbers of the landfill and garage. The third card is optional and may be repeated up to 500 times. It contains a segment number and a section number. The section number indicates the section to which the indicated segment is to be reassigned.

b. Disk Files

Disk file TAPE1 contains segment data and the map distance conversion factor for the overall map. All of the data are read by one binary READ statement. The following is the list used in the READ statement:

```
NSEG,((STG(J,I),J=1,8),DUMMY,DUMMY,STG(NSF,I),I=1,NSEG),AVMD
```

The first word is the count of the segments. The segment data follow, 11 words per segment. After the segment data comes the overall distance conversion factor. The STG array is equivalenced to the ISTG array, so some of the contents in the STG list are integers.

TABLE 3. PHASE3 DATA CARDS

Card	Columns	Format	Contents
1	1-80	8A10	Title
2	1-5	I5	Node number of landfill
	6-10	I5	Node number of garage
The following card is optional and may be repeated up to 500 times.			
3	1-5	I5	Segment number
	6-10	I5	New section number

TAPE1 contains the following data:

```
NSEG         = number of segments
ISTG(1,J)    = street number                          ┐
ISTG(2,J)    = starting node number                   │
ISTG(3,J)    = ending node number                     │
STG(4,J)     = street length, in miles                │
ISTG(5,J)    = number of houses                       │
STG(6,J)     = speed limit, in mph                    ├─ These are repeated
ISTG(7,J)    = number of ways of travel               │  for each segment.
STG(8,J)     = refuse quantity adjustment factor      │
STG(9,J)     = x-coordinate of segment midpoint       │
STG(10,J)    = y-coordinate of segment midpoint       │
ISTG(11,J)   = shape code                             ┘
AVMD         = map distance conversion factor
```

The index, J, corresponds to the Jth segment and is also the segment number. The segment midpoint coordinates are not needed by PHASE3 and are read into variable DUMMY. The map distance conversion factor is not used by the program either, since no maps are drawn.

Disk file TAPE2 contains refuse quantity information and node data. All of the data are read by one binary READ statement. The following is the list used in the READ statement:

NHTOT,TOTREF,KNODES,(NODNUM(I),NBRSEG(I),XNOD(I),YNOD(I),I=1,KNODES)

The first three words are the total number of houses, the total refuse quantity, and a count of the nodes. The node data follow, four words per node.

TAPE2 contains the following data:

NHTOT	= total number of houses or stops	
TOTREF	= total refuse quantity	
KNODES	= number of nodes	
NODNUM(I)	= node number	These are repeated for each node.
NBRSEG(I)	= packed bounding segment numbers	
XNOD(I)	= x-coordinate of node	
YNOD(I)	= y-coordinate of node	

The index, I, corresponds to the Ith node.

File TAPE3 contains pointers to the first and last segments in each section and the vehicle capacity for each section. All of the data are read by one binary READ statement. The following list is used in the READ statement:

NSEG,NTRIP,(SINL(I),SFNL(I),TC(I),I=1,NTRIP)

The first two words are a count of the segments and a count of the sections. The initial segment pointer, the final segment pointer, and the vehicle capacity follow and are repeated for each section.

TAPE3 contains the following data:

NSEG	= count of segments	
NTRIP	= count of sections	
SINL(I)	= sequence of number of first segment in section I	These are repeated for each section.
SFNL(I)	= sequence of number of last segment in section I	
TC(I)	= capacity of vehicle servicing section I	

The pointers refer to the position of the segment numbers on file TAPE4.

File TAPE4 holds the segment numbers in the order in which they are assigned to sections. The segment numbers are read one at a time by a formatted READ statement, using format I6. The numbers are stored in the ISTG array. The segment numbers are grouped by section. All segments in section 1 are listed, followed by all segments in section 2, and so forth. The segment numbers within each section are not necessarily listed in order. Instead, they occur in the order in which they were selected for addition to the section by program PHASE2.

2. SCRATCH FILE

Disk file TAPE7 is used as a scratch file. The data are first written to TAPE7 by a BUFFER OUT statement with parity 1 at statement 86 of PHASE3. The region written extends from the beginning of array NNTF through variable XXX. The saved data include arrays NNTF, DTF, TTF, and NSTF. Each array is a double-subscripted array with second dimension 3. The data in these arrays describe paths from the landfill node to each other node, traveled away from the landfill; paths from the garage node to each other node; and paths from the landfill node to each other node, traveled towards the landfill. Array NNTF contains line numbers of the next node in each path. Array DTF contains distances to or from each node. Array TTF contains travel times to or from each node. Array NSTF contains the segment numbers for the next segments in each path.

The data are saved on disk to free storage for other purposes during the execution of the program. The data are read by a BUFFER IN statement in subroutine EPXP and at statement 390 of the main program. In both cases, paths from the landfill or garage to the section are needed.

3. OUTPUT

Program PHASE3 produces disk and printed output. The descriptions of the final routes are written to file TAPE9. Information is printed to allow the user to verify user-initiated changes in section assignments and to permit debugging if errors should occur.

a. Disk Output

Disk file TAPE9 contains information describing travel and collection for each vehicle. The data are written in card-image form by eight formatted WRITE statements in main program PHASE3. A description of the path from the landfill to the garage is written at the beginning of the file. Two paths follow for each section, one starting at the landfill and one starting at the garage.

Each trip is broken into three pieces. The first piece describes travel from the garage or landfill to the collection region. The second piece describes travel and collection in the collection region. The third piece describes travel from the last collection point to the landfill. There is a header card for each piece. Each header card contains a segment count, the total distance in miles for the piece, the section number, the vehicle capacity, and the amount of refuse to be collected from the section. The header card is followed by one or more cards that describe the path for that piece of the trip. The path is described by repeating three items: a node number, a segment number, and the letter T or C to indicate travel or collection on that segment. The format used for each card image is 8(I5,I4,A1). Up to eight segments can be described on one card image. The number of the node terminating the piece of the trip follows the three items describing the last segment.

The header and path data for the path from the landfill to the garage are written shortly before statement 86 of the main program. The header includes only the number of segments in the path and the length of the path. The headers and path descriptions for the three pieces of each trip are written by six WRITE statements from statement 490 to statement 500 in the main program.

The data on TAPE9 are in card-image form so they may be punched instead of placed on disk. This feature allows the user to modify trips before the data are used by program PHASE4. File TAPE9 can also be copied to the output file for visual examination prior to use.

b. Printed Output

The printed output consists of five sections: a listing of initial section assignments, a listing of changes to section assignments, a summary of segment data, a summary of node data, and a listing of suitable entry and exit points for each section. Samples of each type of printed output are presented in Appendix D.

The problem title is printed at the beginning of the printed output. The segment assignments produced by program PHASE2 follow. The assignments are listed by section in the order in which the segments occur on file TAPE4. Up to 30 segment numbers are printed per line.

Program and user-initiated section changes are printed next. The five columns of information include the segment number, the initial section assignment, the new section assignment, the amount of refuse on that segment, and the type of adjustment (program-adjusted or user-adjusted). The program-initiated changes are printed first. A summary of the vehicle loads and capacities for each trip are printed following the program-initiated changes. The segment number and new section number in the user-initiated changes are read from data cards. The final vehicle capacities and loads for each trip are printed following the user-initiated section reassignments.

The first column of segment data gives the segment number. The second column, headed NSTR, gives the street number for the segment. Columns NN1 and NN2 give the starting and ending node numbers. Column LEN gives the segment length in miles. Column NH gives the number of houses on the segment. Column SPD gives the speed limit on the segment. Column NWAY gives the number of directions of travel allowed on the segment. Column RQF gives the refuse quantity adjustment factor. Column NSCT gives the final section assignment of the segment. Segments containing no refuse have a zero as their section assignment. The last column, headed SF, gives the shape code in octal.

The node numbers of the landfill and garage are given at the beginning of the node data list. The first column, headed LINE, is the line number of the node. The second column, headed NODE, is the node number. The columns headed NNFD, NNTD, and NNFG give the line numbers of the next node on the shortest path (1) to the landfill traveling away from the landfill, (2) to the landfill traveling toward the landfill, and (3) to the garage traveling away from the garage. Columns NSFD, NSTD, and NSFG give the segment numbers of the next segment in each of the three types of paths. Columns DFD, DTD, and DFG give the distance of the node from the landfill or garage for the three types of paths. The travel times, in minutes, are given in the next three columns, headed TFD, TTD, and TFG. Data for paths traveled toward and away from the

landfill are different only when one-way streets cause the travel paths to be different. The column headed NUMNBR gives a count of segments bounding the node. The remaining six columns, headed NEIGHBORING SEGMENT NUMBERS, give the numbers of the segments bounding the node. Some of the columns will be blank when a node has fewer than six neighboring segments.

Entry and exit nodes are listed for each section. The line starting FROM DUMP gives the numbers of nodes that are suitable entry points to the section for a vehicle starting at the landfill. The line starting FROM GARAGE gives the numbers of nodes that are suitable entry points to the section for a vehicle coming from the garage. The line starting TO DUMP gives the numbers of nodes that are suitable points for leaving the section to travel to the landfill. From one to ten node numbers may be listed on each line. Variable JMAX in the main program limits the number of entry points, and variable KMAX limits the number of exit-point selections. At present, both JMAX and KMAX are set to 4.

If file TAPE9 is copied to the output file, the printout will follow the entry- and exit-point listing.

(The reverse of this page is blank.)

SECTION V
PROGRAM REQUIREMENTS

1. SYSTEM

Program PHASE3 is written entirely in FORTRAN IV. The program runs on a CDC 6600 computer using a SCOPE 3.4.4 operating system.

Nine obvious types of computer-dependent coding occur in program PHASE3 and its subroutines. Main program PHASE3 assumes a 130-character output line, 10 characters per word, and a 60-bit word. System function SHIFT is used in the main program and in subroutines ADJUST, TREE, CON2ST, CONNECT, DISCON, and CLOSE1. Masking operations are used in subroutines TREE, CON2ST, CONNECT, DISCON, and CLOSE1. Asterisk-bounded text is used in format statements in the main program and in subroutines ADJUST, PRNPCS, TRACE, TREE, FNDPTH, CONNST, DISCON, EPXP, SOLV, and TRAVEL. An Iw.n format specification is used in subroutines PRNPCS and EPXP and in the main program. Multiple replacement statements occur in the main program and in subroutines MOVE3, CON2ST, FNDPTH, CONNECT, EPXP, CLOSE1, GENDM, TRAVEL, and OPTPATH. A dollar sign is used to separate FORTRAN statements in the main program and all subprograms except subroutine GENDM, function IFIND, and function CUMDIS.

More subtle types of machine dependencies may exist, according to the machine used.

2. STORAGE

The core requirement is slightly less than $123{,}000_8$ words. Exactly 3,601 words of data are written to file TAPE7. The maximum peripheral storage used by file TAPE9 should not exceed 33,800 words.

3. TIME

The running time for PHASE3 depends on the number of sections and the number of entry and exit points. Since the number of entry and exit points cannot be determined by the user before the program is run, a precise estimate of running time cannot be given. An upper limit on the CP time, in minutes, should be 0.36 times the number of sections. Very rough limits on the IO and PP times are 0.06 minutes IO time per section and 0.12 minutes PP time per section.

SECTION VI
PROGRAM LIMITATIONS

The only new limitation imposed upon the program user by program PHASE3 is a limit of 500 on the number of section reassignment cards. More cards are permitted, but since there can be no more than 500 segments, there would be no need for additional reassignment cards.

Limitations imposed on the map description by the dimensions in PHASE3 are 500 segments and 300 nodes. The number of trips is limited to 50. These limitations should have been observed when programs RCINPT and PHASE2 were run. The segments connected to the landfill and garage must not include houses.

Four limitations exist which may cause problems that the user usually cannot anticipate: (1) One-way streets may prevent the program from finding a travel path for a trip, (2) storage has been dimensioned to allow pairing of at most 32 odd nodes, (3) array dimensions limit each of the three pieces of each trip to 100 segments, and (4) array dimensions limit the total number of steps in the three pieces to 200. Corrective procedures for dealing with these limitations are discussed following the corresponding error messages in the next section.

(The reverse of this page is blank.)

SECTION VII
ERROR MESSAGES AND CORRECTIVE ACTION

1. STOP 30

Type: Fatal.
Source: Subroutine CONNECT.
Location: Day file.
Meaning: No segments have been found in section NTRIP by subroutine CONNECT. The data on file TAPE3 or on file TAPE4 may be incorrect, or the user may have removed all segments from the section by using section reassignment cards.
Action: If the user has intentionally removed the section, the STOP 30 statement in subroutine CONNECT should be replaced by a RETURN statement. If the problem has been created unintentionally, the user should determine whether files TAPE3 and TAPE4 correspond to the current problem. If all segment-related data are correct, the error may be completely unrelated to the section assignments. Other indicated errors should be corrected.

2. STOP 111

Type: Fatal.
Source: Subroutine CONNECT.
Location: Day file.
Meaning: The number of a starting or ending node in the segment data has not been found in array NODNUM.
Action: Verify that the segment data and node data permanent files correspond to the same problem. Correct any other errors indicated by the program. If the problem persists, a core dump of the program will be necessary. Variable NNN(J) in subroutine CONNECT should be found. If the number is a valid node number, then it should also appear in array NODNUM. If it does not appear in array NODNUM, a programmer should determine why it is not there. If NNN(J) is not a valid node, the programmer should determine how it was obtained from the ISTG array.

3. STOP 201

Type: Fatal.

Source: Subroutine CLOSE1.

Location: Day file.

Meaning: A node number in the segment data array could not be found in array NODNUM.

Action: Verify that the node and segment data permanent files belong to the same problem. Correct any other indicated error. If the problem persists, it may be caused by an unrelated program logic error. A full core dump may be necessary to find the cause.

4. STOP 230

Type: Fatal.

Source: Subroutine CONNECT.

Location: Day file.

Meaning: More than NPIECE pieces were found in section NTRIP in the loop through statement 230 in subroutine CONNECT. The piece count, NPIECE, is probably in error.

Action: The computation of NPIECE should be examined for errors. NPIECE is set to 1 at the beginning of subroutine CONNECT and is incremented following statement 70. If no error is found in the computation, the temporary section assignments in the ISTG array should be examined to find the cause of an invalid piece number in these data.

5. STOP 270

Type: Fatal.

Source: Subroutine CONNECT.

Location: Day file.

Meaning: Piece number NTRIP could not be found in the NNFN array in the loop through statement 280 in subroutine CONNECT. An error in program logic or a machine malfunction probably has occurred.

Action: Any other indicated errors should be corrected. Variable NTRIP and the first NPIECE entries in the NNFN array should be examined, either from a dump or from a user-inserted print statement. If NTRIP is not present in the NNFN array, the error has occurred between statements 240 and 280. A programmer should find and correct the omission.

6. STOP 501

Type: Fatal.

Source: Subroutine CLOSE1.

Location: Day file.

Meaning: A node in section NTRIP has no neighboring segments in that section. Either the node should not be in array NODPTR, or one of its *bounding* segments should be in the section.

Action: Verify that the node data and segment data permanent files correspond to the same problem. If the error is not in the original node or segment data, a program logic error is probably the cause. The selection of nodes in the loop through statement 220 in the main program should be reexamined.

7. STOP 1401

Type: Fatal.

Source: Subroutine CLOSE1.

Location: Day file.

Meaning: More than 32 steps (segments) are needed to return to the section from an order-one node at the end of a one-way street.

Action: If the map description is correct, the segment should be reassigned to a different section. More likely, the map description is incorrect or consists of two separate regions connected by a one-way street. The description should be corrected, or additional segments should be added to the map description to connect the two pieces.

8. STOP 1601

Type: Fatal.

Source: Subroutine CLOSE1.

Location: Day file.

Meaning: The line number, LF, of the node at which a path returns to the section cannot be found in the NODPTR array in the loop through statement 160. The program logic is in error, probably in the selection of the return path.

Action: The path data in arrays IPATHS and IPATHN should be examined. The arrays are filled in the loop through statement 110. The type of error in the path should indicate the probable source of the error.

9. NODE nnnnn IS NOT IN THE NODE TABLE
CORRECT THE DUMP OR GARAGE NUMBER
JOB TERMINATED

Note: nnnnn is a number from 1 to 99999.

Type: Fatal.

Source: Subroutine TREE.

Location: This message may be printed in one of three places: following the section reassignments, following the node data, or following the entry- and exit-point listing.

Meaning: If the message is printed immediately following the section reassignments, then the node number of the landfill or garage specified on the second data card does not match any node number in the node data. Otherwise, a node number from a starting or ending node in the segment data has not been found in the node data.

Action: If the error message is printed immediately following the section reassignments, the node number of the landfill or garage on the second data card should be corrected. Otherwise, verify that the segment and node data permanent files correspond to the same problem.

10. IMPROPER LINE POINTER USED IN TRACE
JOB TERMINATED

Type: Fatal.

Source: Subroutine TRACE.

Location: This message may occur in one of two places: immediately following the section reassignments, or following the entry- and exit-point listing.

Meaning: Either the line number, L, is outside the range of permissible node line numbers (1 through KNODES), or the number of segments in the path, NSG, exceeds 200.

Action: If the message is immediately after the section reassignments, the problem is in the trip from the garage to the landfill. If the

message follows the entry- and exit-point printout, the problem may be in the path between any two nodes. The numbers of the nodes that start and end the path are indicated in the error message as variables NSTART and NSTOP. If the number of segments, NSG, exceeds 200, the user should compare the path to the input maps. If more than 200 steps are required, the dimensions of arrays IPS and IPN in the main program should be increased. If the path is incorrect, or if the line pointer, L, is improper, verify that the node and segment data permanent files correspond to the same problem. If the error persists after all other errors have been corrected, the user must find the cause. A core dump is provided after the error message.

11. NODE nnnnn DOES NOT CONNECT TO THE α

CORRECT THE DUMP OR GARAGE NUMBERS IF INCORRECT OR ELSE
CORRECT THE MAP DESCRIPTION CARDS AND RERUN THE INPUT AND
SECTIONING PROGRAMS

JOB TERMINATED

Note: nnnnn is a node number from 1 through 99999.

α is either DUMP, GARAGE, or DUMP OR TO THE GARAGE.

Type: Fatal.

Source: Main program PHASE3.

Location: Following the node listing.

Meaning: Self explanatory.

Action: Self explanatory.

12. MORE THAN mm STEPS NEEDED TO CONNECT PIECES ppppp AND ppppp

IF THIS IS CORRECT, INCREASE THE FIRST DIMENSION OF IPATH IN
COMMON BLOCK TEMSTG, SUBROUTINES FNDPTH AND CONNST

JOB TERMINATED

Note: In the current version of the program, mm will be 50. ppppp represents a piece number. The tens and units digits give the section number, and the thousands and ten-thousands digits give the piece number. Additional printout will give the node and piece numbers of the end points and the first 15 segments in the path.

Type: Fatal.

Source: Subroutine FNDPTH.

Location: Following the node data listing.

Meaning: Self explanatory.

Action: If a path shorter than 50 steps exists between the two pieces, an error in the program logic is present, probably in subroutine FNDPTH. A programmer should seek the error.

13. NO CONVERGENCE IN CONNST

Type: Warning.

Source: Subroutine CONNST.

Location: Following the node data listing.

Meaning: No paths could be found between some or all of the pieces of the current section. The connections may be one-way streets, or the map description may be in error. An error may exist in the logic in subroutine CONNST.

Action: *Obtain a core dump. The value of variable ITRIP in subroutine* CONNECT gives the number of the section containing the problem. Examine the pieces of this section, using a map from program PHASE2 and the section assignments in the segment list. If errors are found in the map description, they should be corrected. If one-way streets are the only paths between pieces of the section, the streets must be made two-way streets. The paths generated by PHASE3 on file TAPE9 should be punched and adjusted by hand to conform to the one-way streets. If no problems are found in the map description and if no one-way problems exist, a programmer should examine subroutine CONNST for logic errors.

14. CONNST ITER nn

Note: In the current version of the program, nn will be 11.

Type: Warning, but the results from PHASE3 cannot be used by PHASE4 until the problem has been corrected.

Source: Subroutine CONNST.

Location: Following the node data listing.

Meaning: Subroutine CONNST has been unable to connect all of the pieces of a section in 10 iterations. Since at least five pieces are connected by each iteration, there are at least 50 pieces to the section

requiring more than two step connections, or the logic in subroutine CONNST is in error.

Action: Verify the map description as for error type 13. If the section consists of more than 50 pieces, section reassignment cards should be used to make the section more compact. If the subroutine logic is in error, a programmer should seek the error.

15. I = ddd NODE = nnnn

Note: ddd represents a subscript from 1 through 100, and nnnn represents a node number. A table of node pointers and node orders follows the message, and a core dump follows the table.

Type: Fatal.

Source: Subroutine DISCON.

Location: Following the node data listing.

Meaning: An end-point node of a segment could not be found in the node data. Variable I indicates the subscript of the node in the node pointer and node order arrays. Variable NODE gives the node number.

Action: Verify that the node and segment data permanent files correspond to the same problem. Correct any other indicated errors. If the error persists, a programmer should examine subroutine DISCON for logic errors.

16. LINE NODE NXND NXSG DIST TIME SECT

Note: A table of data follows the indicated column headings. A core dump follows the table.

Type: Fatal.

Source: Subroutine CLOSE1.

Location: Following the entry- and exit-point listing.

Meaning: A break has occurred in a path generated by subroutine TREE from an order-one node back to the current section. A malfunction in subroutine TREE is the most probable cause.

Action: All other errors should be corrected. If the problem persists, the path should be examined for possible map problems. The pointer to the starting node of the path can be found in the dump as NODPTR(K), where K is a variable in subroutine CLOSE1. The number of the final

node of the path is variable NORG in subroutine CLOSE1. To trace the path, find the line number of the starting node in the column headed LINE. The number of the node is in the column headed NODE. The next segment in the path is in the column headed NXSG. The section assignment of the segment is in the column headed SECT. The line number of the next node in the path is in the column headed NXND. Columns DIST and TIME give the distance and time from the node in column NODE to node number NORG. The break in the path occurs where NXND refers to a line that has not been printed, or where NXSG contains a zero, in the table that comprises part of this error message.

17. THE NUMBER OF ODD NODES, nnn, EXCEEDS mmm
EITHER MODIFY THE MAP TO PAIR OR REMOVE ODD NODES OR ELSE
SEE THE PROGRAM DESCRIPTION MANUAL FOR INSTRUCTIONS
NODE/ORDER

Note: Up to 10 node/order pairs per line are printed following the NODE/ORDER heading. A list of segments in each piece of the section follows this tabulation.

Type: Warning, unless error message 20 follows.

Source: Main program PHASE3.

Location: Following the entry- and exit-point listing.

Meaning: The section contains more odd nodes than the program is dimensioned to handle.

Action: No action is necessary unless error message 20 follows this message. If error message 20 follows, there are two alternatives: either modify the section, or increase the dimensions of arrays used in odd-node pairing. The segment list will indicate which segments are in the section. The node and order listing will indicate which nodes are odd nodes. At two of the nodes, an even number of segments will meet; these are the entry- and exit-point nodes. The section description should be modified if section reassignment cards can be used to add or remove segments in such a manner that the number of odd nodes is reduced to the maximum (mmm).

If the dimensions used in odd-node pairing are to be increased, changes will be necessary in the main program and in subroutines SELORD, SOLV, NEXTM, and PATH. The dimension of array IPART should be increased to the number of odd nodes in COMMON block PART in the main program and in subroutines SOLV, NEXTM, and PATH. The dimensions of arrays ISTP and JSTP should be increased to the number of odd nodes in COMMON block STP in subroutines SOLV, NEXTM, and PATH. The second and third dimensions of arrays ISTPO and ISTPN should be increased to the number of odd nodes in subroutines NEXTM and PATH. In subroutine NEXTM, both dimensions of arrays MTXN and MTXO should be increased to the number of odd nodes. In subroutine SOLV, the last two dimensions in each array in COMMON block TEMSTG should be increased to the number of odd nodes. If the dimensions are increased to allow more than 36 odd nodes, the dimension of array Z in COMMON block TEMSTG in the main program should be increased to 4000 less than six times the square of the number of odd nodes. In the main program, the dimension of array ONDMTX should be increased to the square of the number of odd nodes. In a DATA statement, the value of MAXODD should be increased to the number of odd nodes.

18. NCH= ddd QUITTING

Note:	ddd will equal 20 in the current version of the program.
Type:	Warning.
Source:	Subroutine SOLV.
Location:	Following the entry- and exit-point listing.
Meaning:	The odd-node pairing improvement did not converge in 20 iterations. A one-way street is probably present on a path that pairs two odd nodes.
Action:	No action is necessary unless error message 20 follows this message. If error message 20 appears, either the one-way street must be changed to a two-way street in the data to program RCINPT, or the odd nodes must be made even nodes using section reassignment cards. If the one-way street is made a two-way street, the TAPE9 output from PHASE3 should be modified if final travel on that street is in the wrong direction.

19. KS ARRAY

Note: A table of 100 numbers, 20 per line, will follow this heading. The arguments to subroutine TRAVEL and data describing the segments in the current section will be printed on the following page. The data are terminated by the message TRAVEL PATH CANNOT BE FOUND. RECHECK NETWORK FOR THIS TRIP.

Type: Warning.

Source: Subroutine TRAVEL.

Location: Following the entry- and exit-point listing.

Meaning: The trial-and-error search for a travel path in the collection region has been unsuccessful in subroutine TRAVEL. There are three possible causes: (1) an error in the map description, (2) one-way streets causing problems in the odd-node pairing, or (3) a logic error somewhere in the program.

Action: No action is necessary unless error message 20 follows this message. If error message 20 follows, the data printed with this message can be used to determine the type of problem.

The KS array indicates the number of times the subroutine has backed up to each step in the path during the trial-and-error search. Subroutine TRAVEL stops the search when it has backed up to a point 101 times. All of the path up to this point indicates the current status of the travel path. The nodes and segments in the path are on the following page of printed output under the column headings IPN and IPS. All of the segments in the section are listed following the heading TRAVEL ARGUMENTS. Column headings indicate the starting and ending node numbers. Column NSG gives the segment numbers. Column NWAY indicates the number of ways of travel on the street. The column headed TIMES indicates the number of times the street is available for traversal.

The path is to start at the node indicated by ISTART and end at the node indicated by ISTOP. The program user should attempt to find a travel path manually, using these nodes and segments, in order to determine the type of problem encountered by subroutine TRAVEL. If one-way streets are causing the problem, they should be made two-way streets in the map description to program RCINPT.

The output from PHASE3 (file TAPE9) should be adjusted if travel on these streets is in the wrong direction. If it is impossible to find a travel path manually, an error probably exists in the selection of the segments in the section.

20. NO PATH EXISTS FOR THE TRIP STARTING AT THE α

Note: α will be either GARAGE or LANDFILL.

Type: Fatal.

Source: Main program PHASE3.

Location: Following error messages after the entry- and exit-point listing.

Meaning: No travel path could be found in the current collection region. Warning message 17, 18, or 19 will precede this error message.

Action: The corrective action is described under the appropriate warning message.

21. I=i LINE=j LIM=m NBRS=n

Note: A table of line numbers, node pointers, and node orders follows the message.

Type: Fatal.

Source: Subroutine DISCON.

Location: Following the node data listing.

Meaning: NODPTR(i) points to a node, NODNUM(j), which should be in the section, but none of the m segments bounding the node are in the section. The segment numbers are packed into each ten bits of the octal number n. The node and segment numbers might be from different problems, or the node pointers in array NODPTR may have been overwritten.

Action: Verify that the node and segment data correspond to the same problem. If the number of nodes in the section (the line number of the last line in the table gives the number of nodes in the section) exceeds 100, the node pointer array has been partially overwritten by the node order array. In this case, the dimensions of arrays NODORD, NODORDS, NODPTR, and NODPTRS can be increased to 200 in the main program. If there are more than 200 nodes in the section, the map description to program RCINPT must be simplified or the problem cannot be run.

If the number of steps to or from the section exceeds 100, or if the total number of steps from the garage or landfill to the section and back to the landfill exceeds 200, the route data on file TAPE9 will be incorrect. No error message is given.

The number of steps in each trip piece can be increased to 200 by increasing to 200 the dimensions of arrays ISFS, ISTS, NDFS, NDTS, NODORD, NODORDS, NODPTR, and NODPTRS in the main program. The total number of steps can be increased to 600 by increasing to 600 the dimensions of arrays ISSV, NDSV, and CORT in the main program. Increases beyond these numbers of steps require extensive program modifications and should be considered beyond the capabilities of the program.

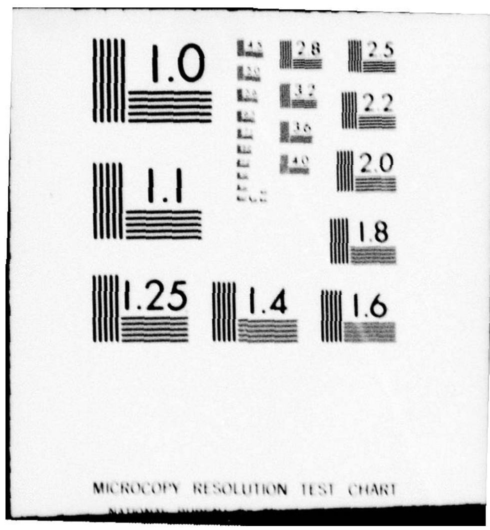
SECTION VIII
RECOMMENDED CHANGE

In subroutine *DISCON*, *an error message* will *not* print node numbers from 10000 to 99999 because of an inadequate field width specification. The first occurrence of I4 in format 45 should be changed to I5.

(The reverse of this page is blank.)

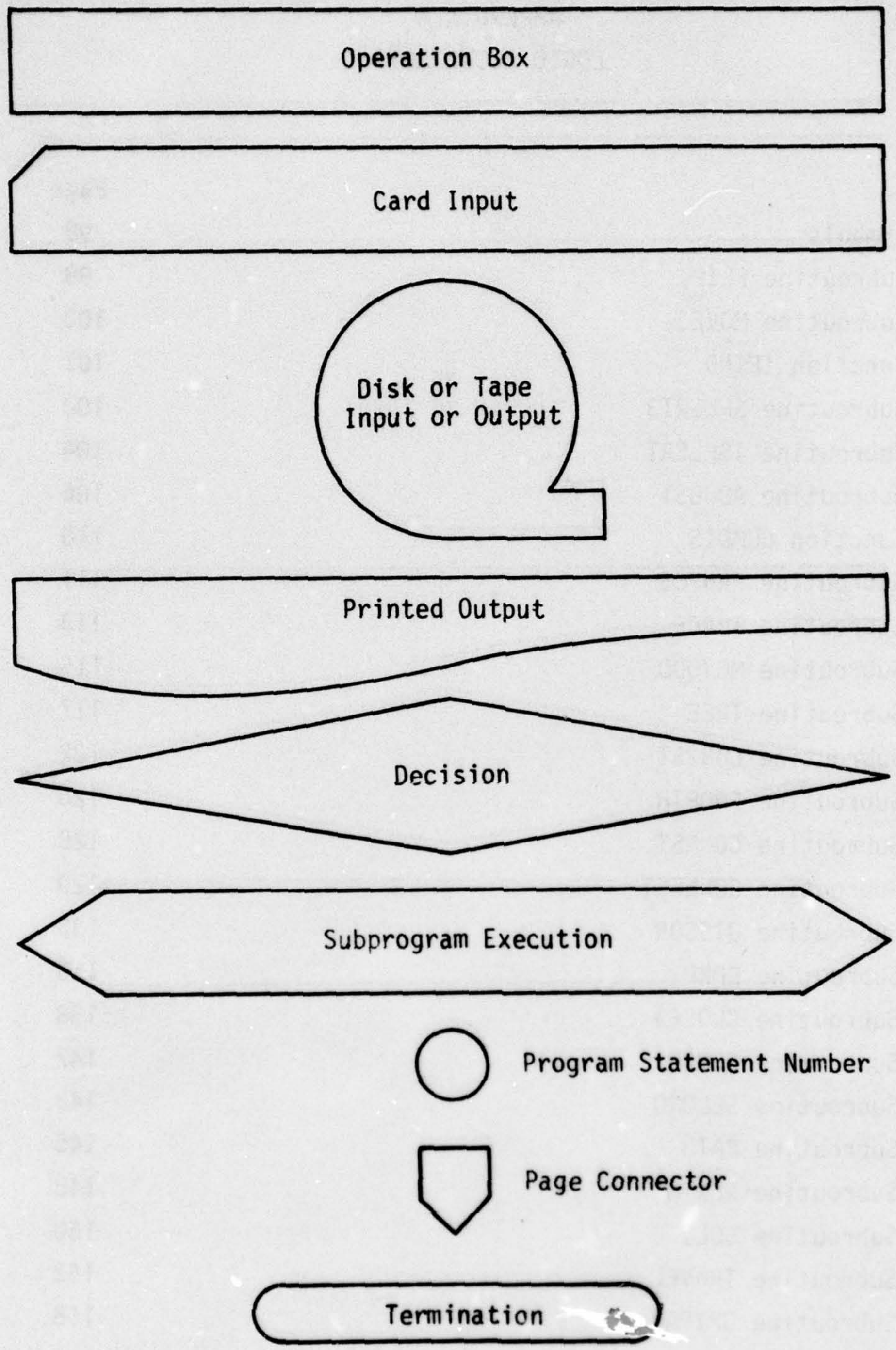
AD-A060 986 NEW MEXICO UNIV ALBUQUERQUE ERIC H WANG CIVIL ENGINE--ETC F/G 13/2
AIR FORCE REFUSE-COLLECTION SCHEDULING PROGRAM DESCRIPTION. VOL--ETC(U)
JUN 78 H J IUZZOLINO F29601-76-C-0015
UNCLASSIFIED CERF-EE-21 CEEDO-TR-78-23-VOL-3 NL

2 OF 3
AD
A060 986

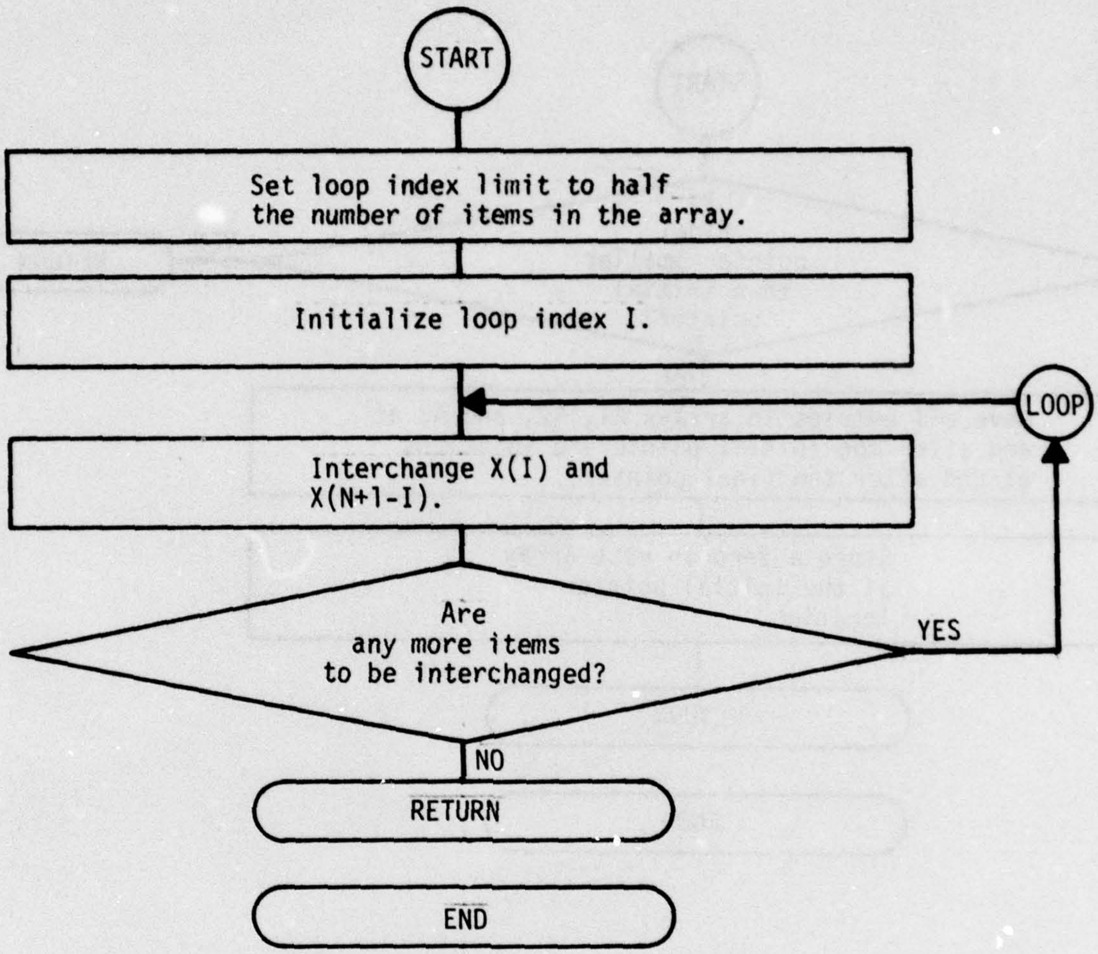

1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART

APPENDIX A

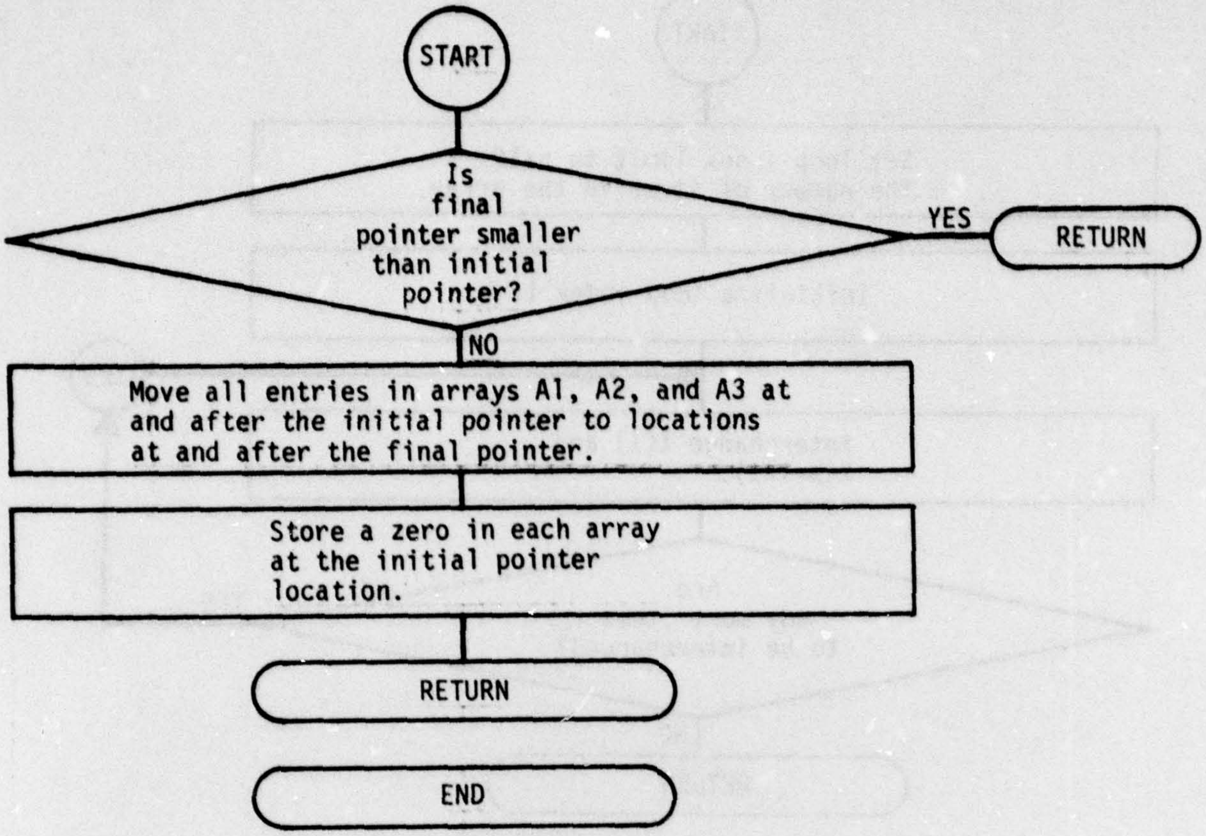
LOGIC FLOWCHARTS



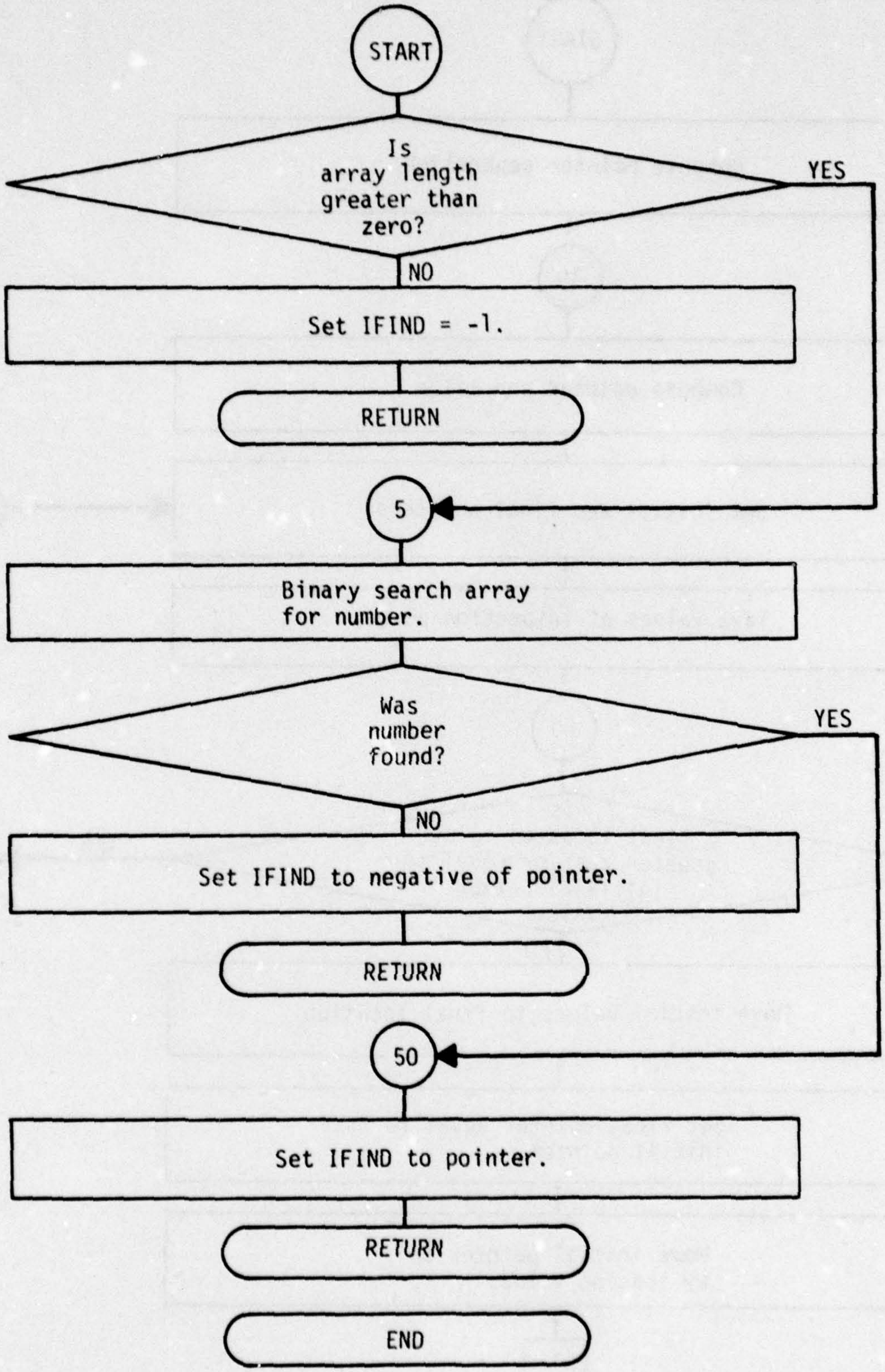
PROGRAM FLOWCHART SYMBOLS



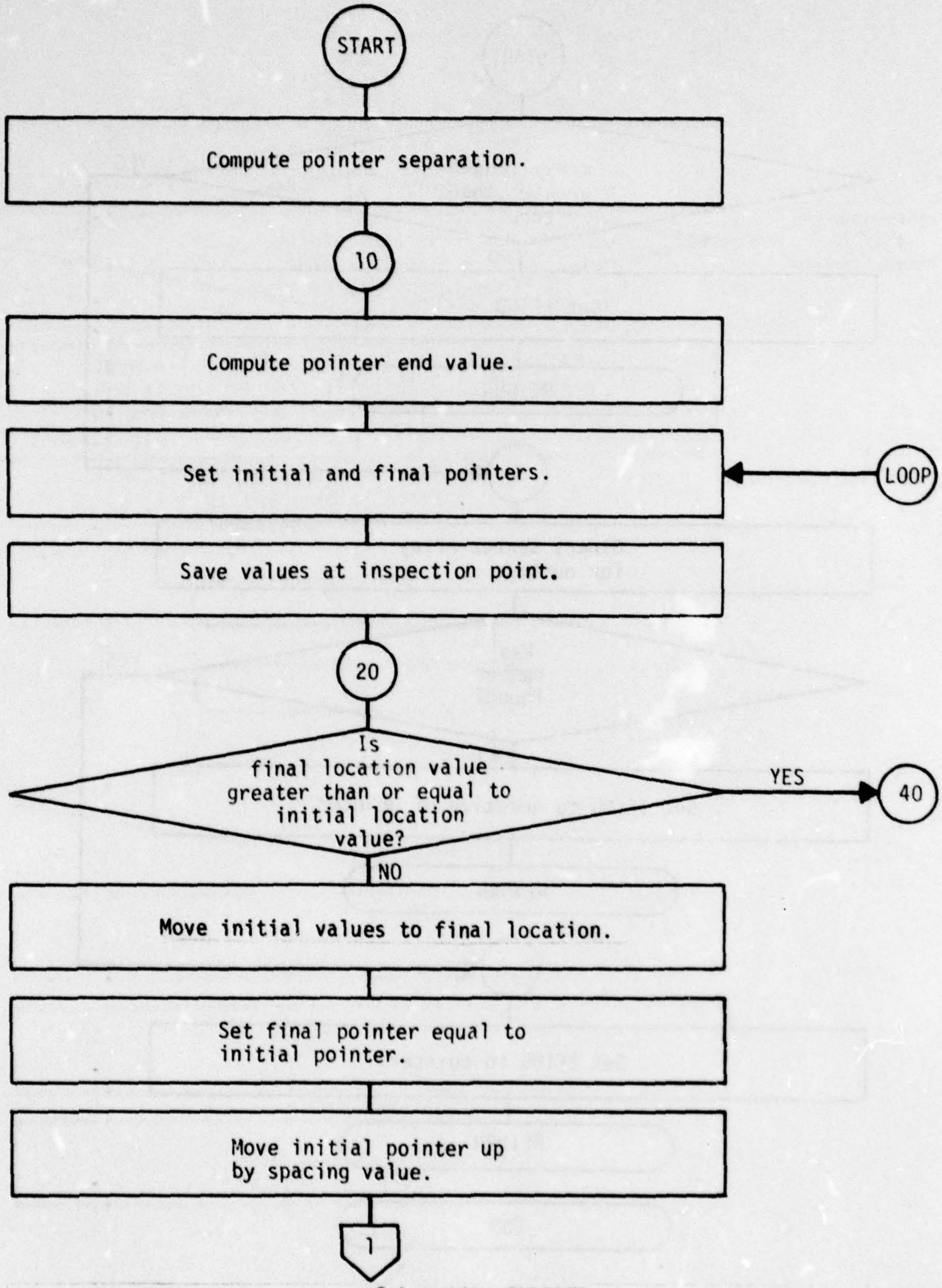
Subroutine FLIP



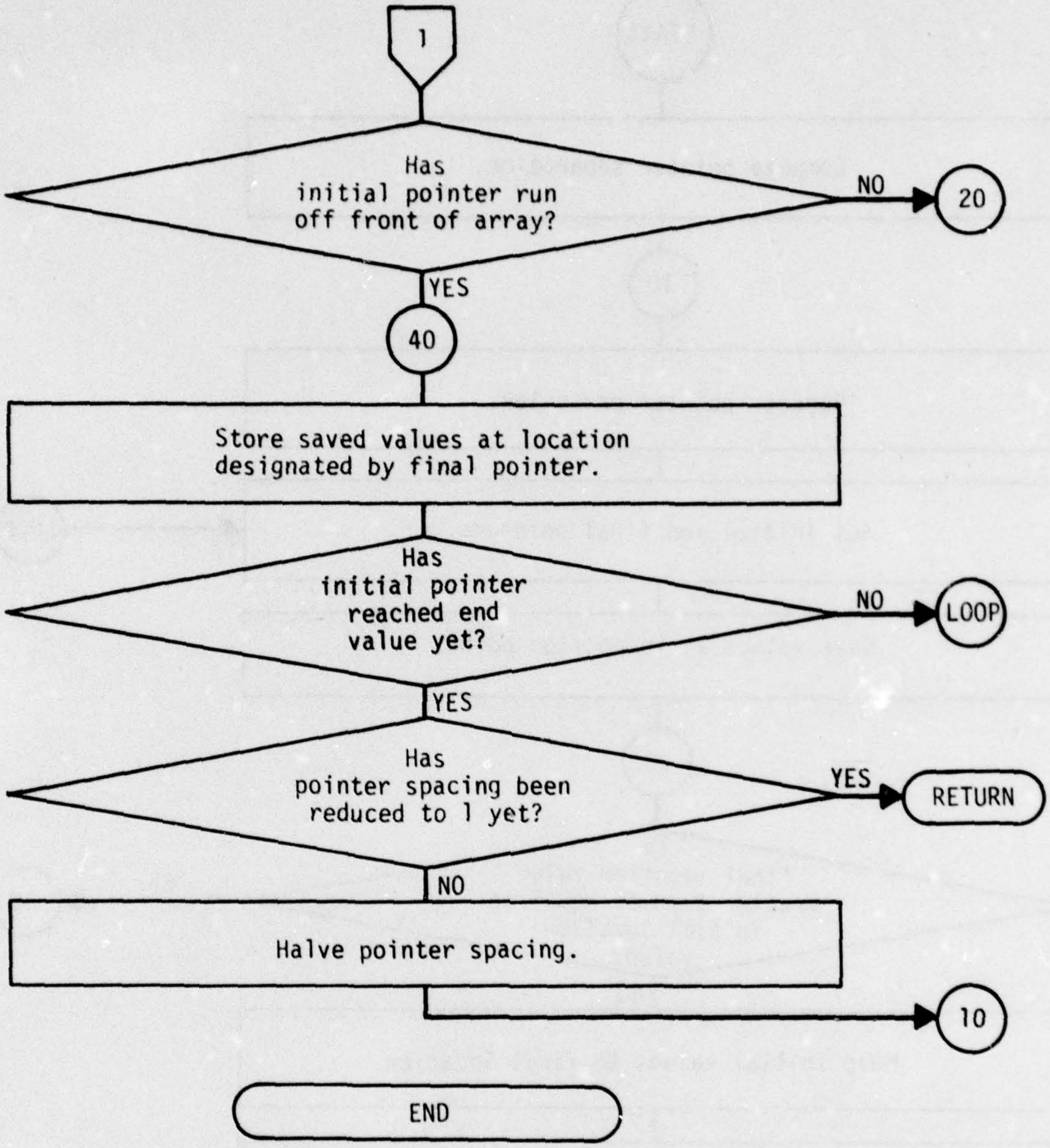
Subroutine MOVE3



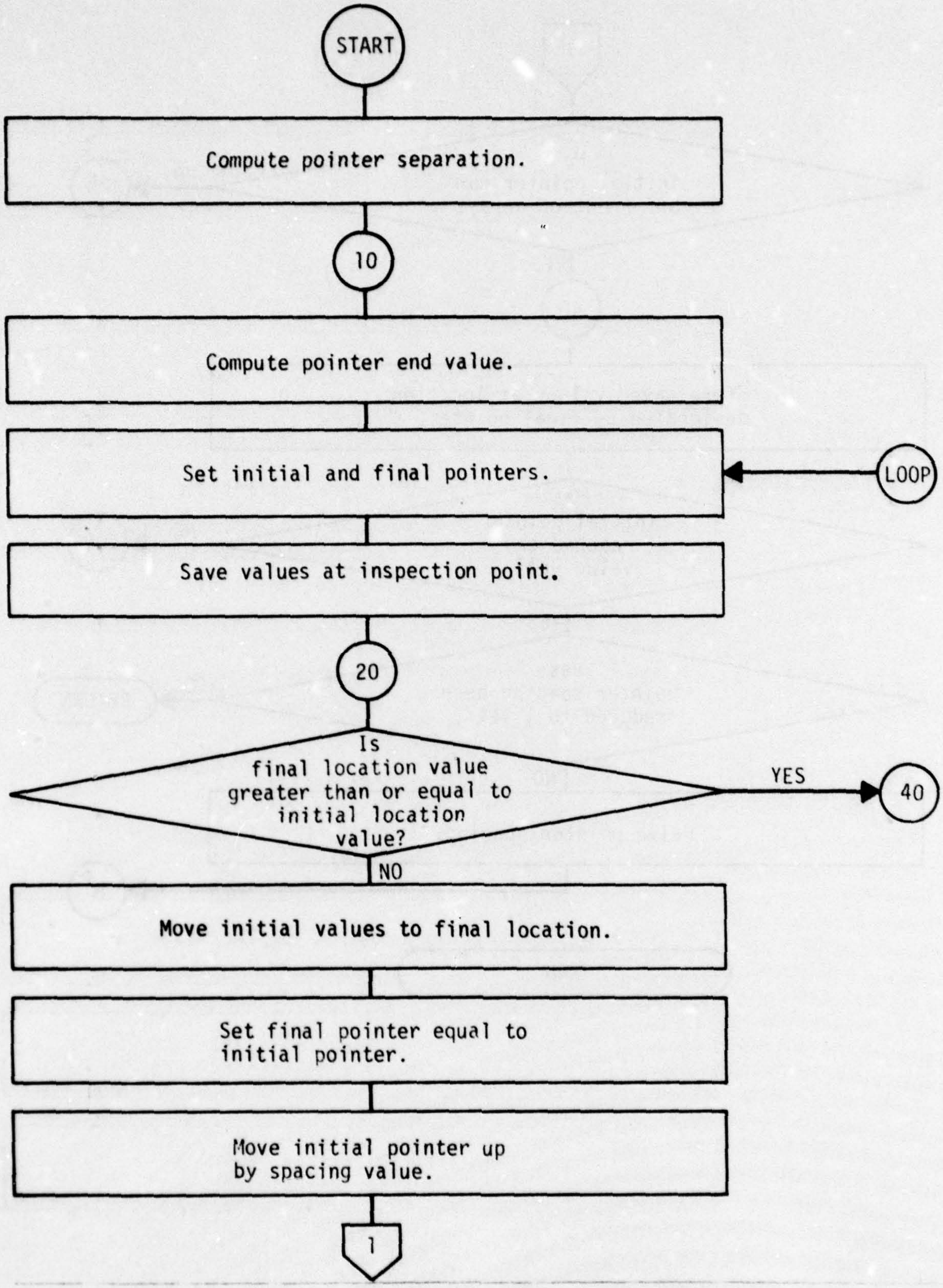
Function IFIND



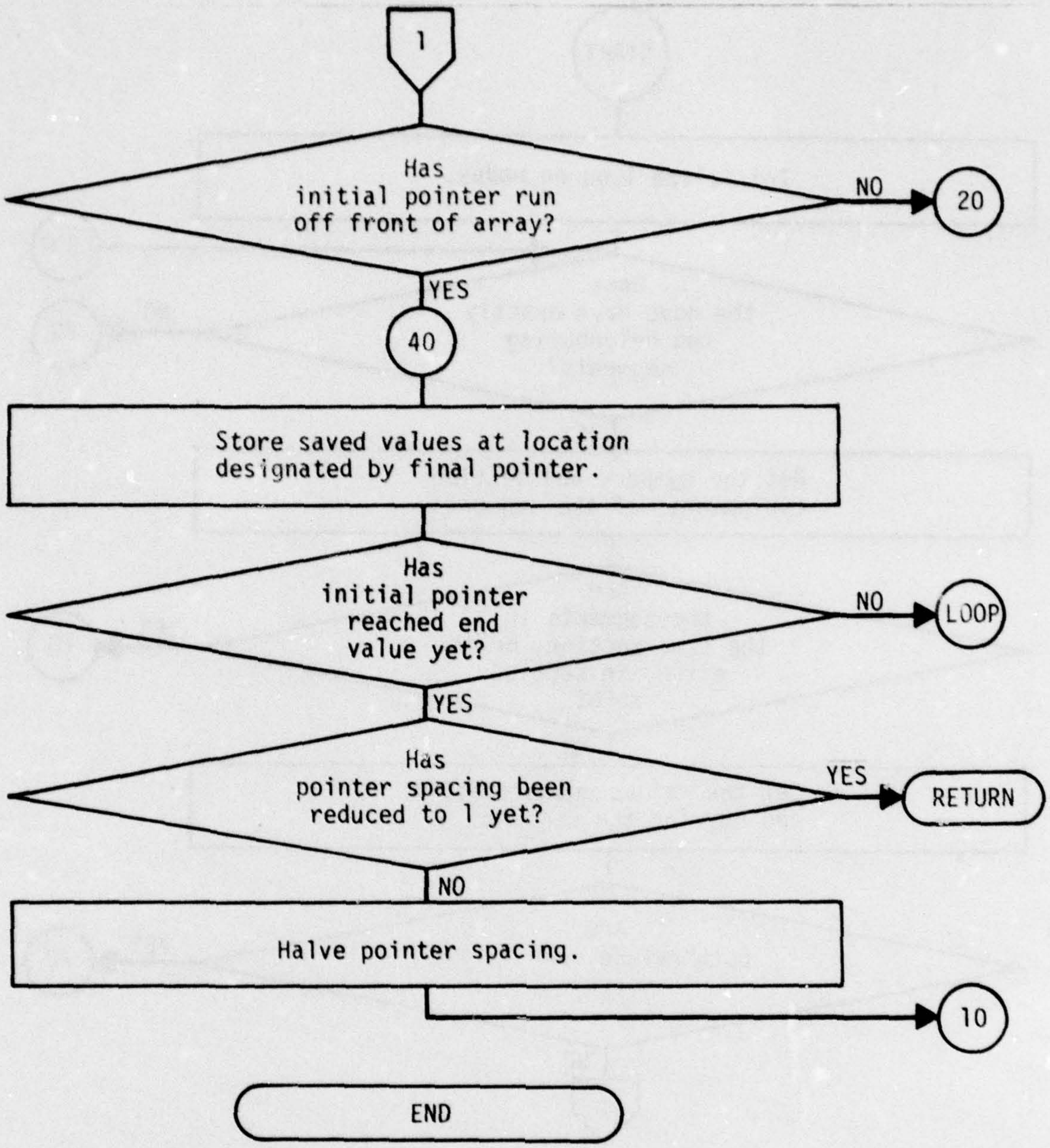
Subroutine SHLSRT3



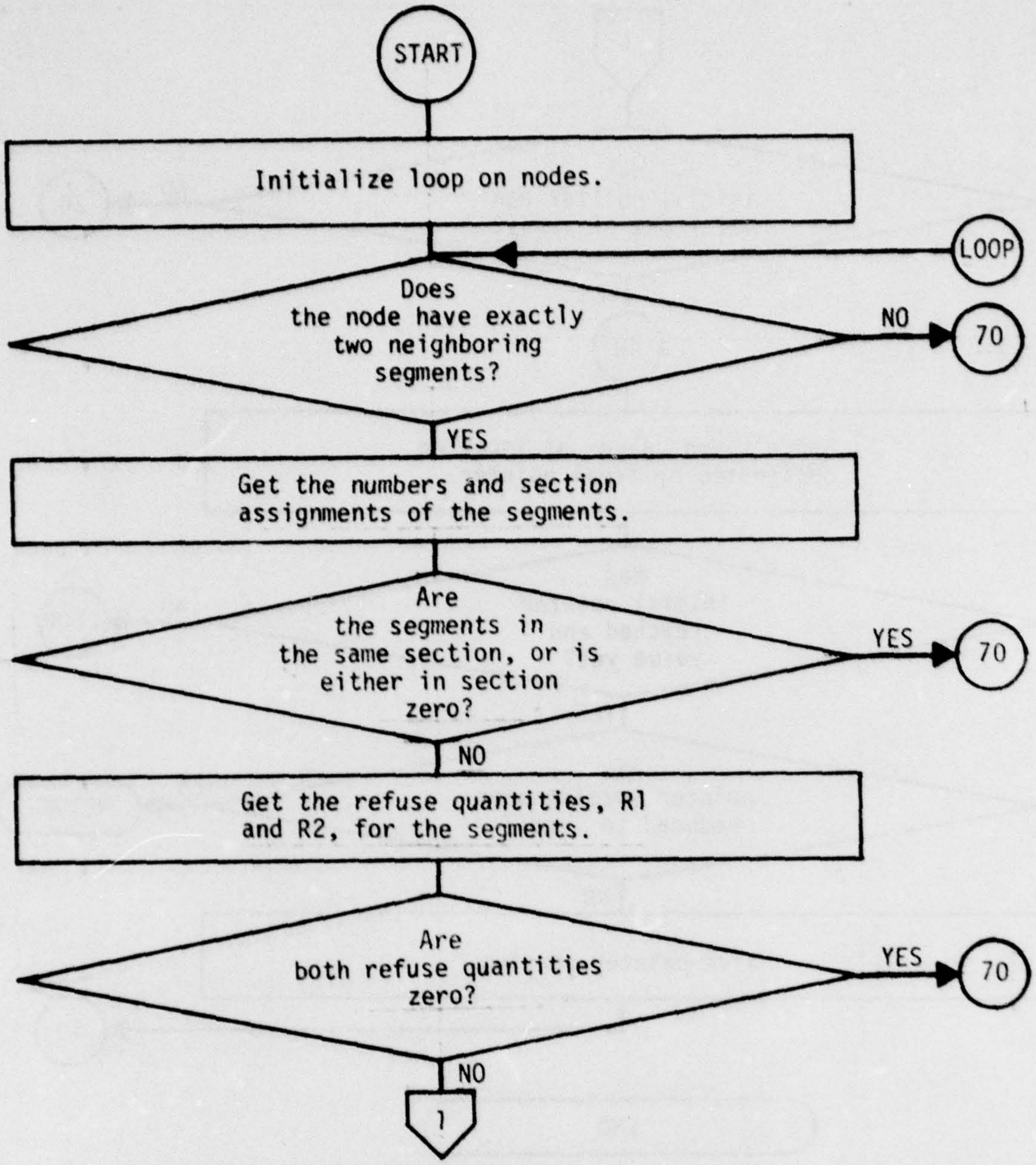
Subroutine SHLSRT3



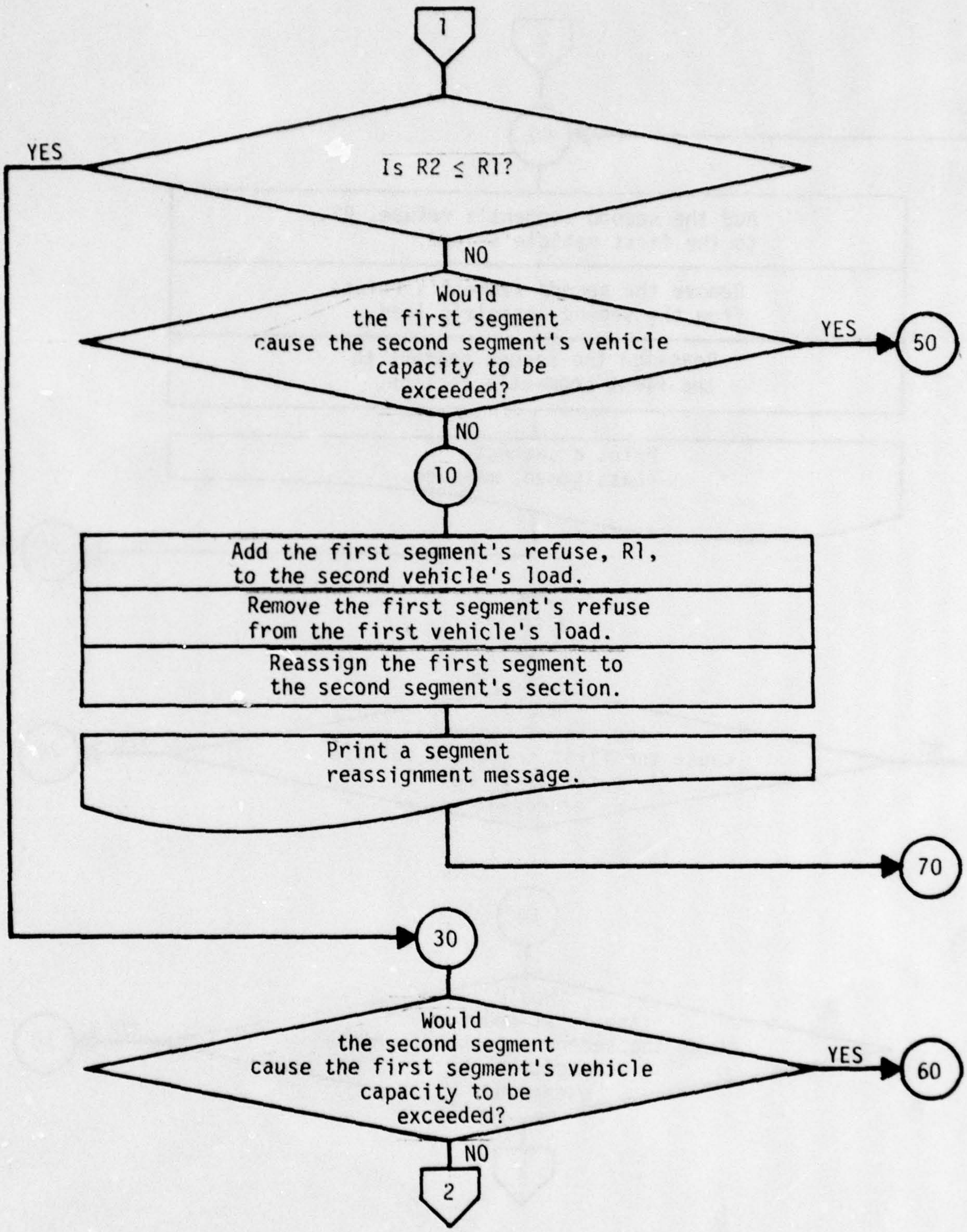
Subroutine ISHLSRT



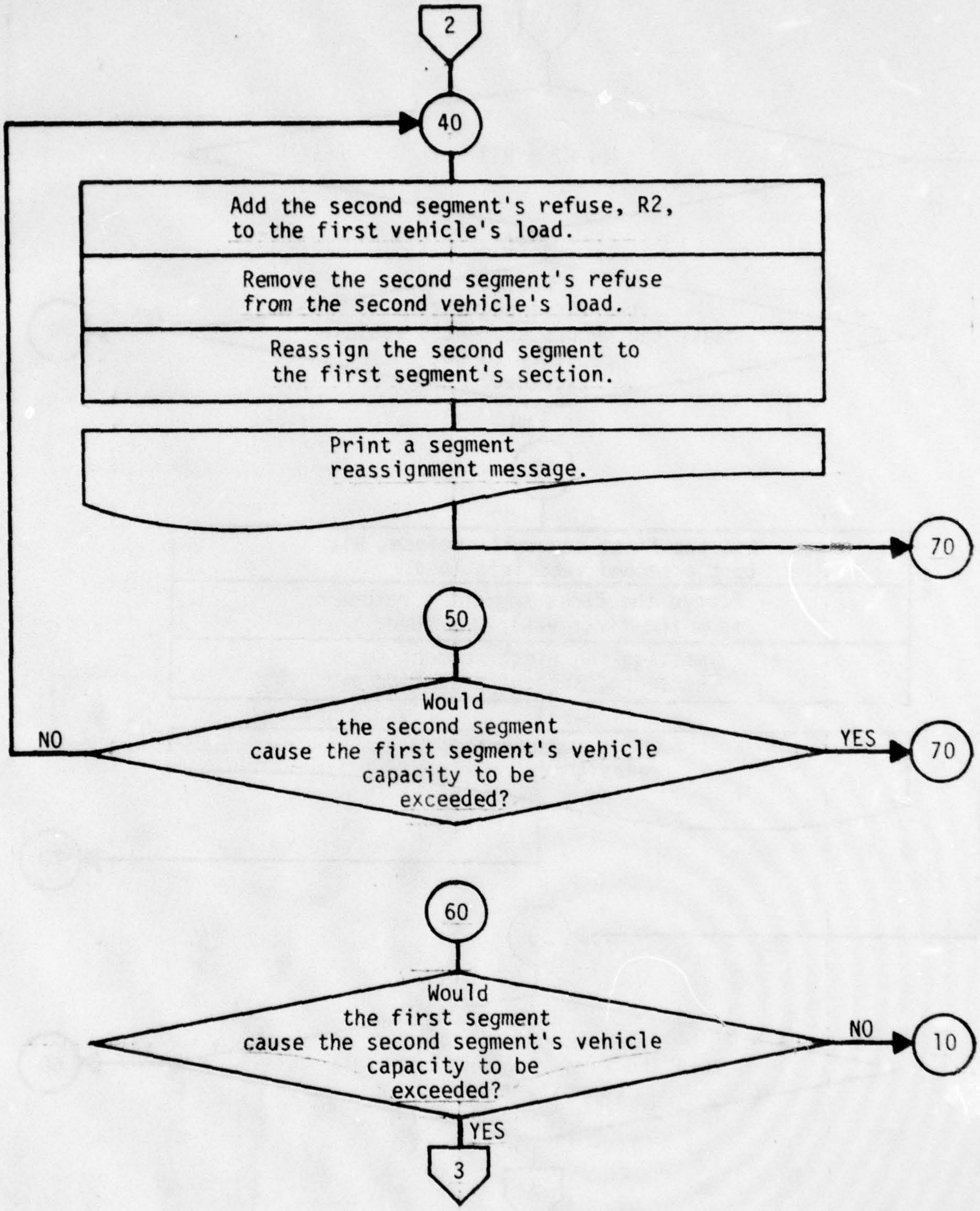
Subroutine ISHLSRT



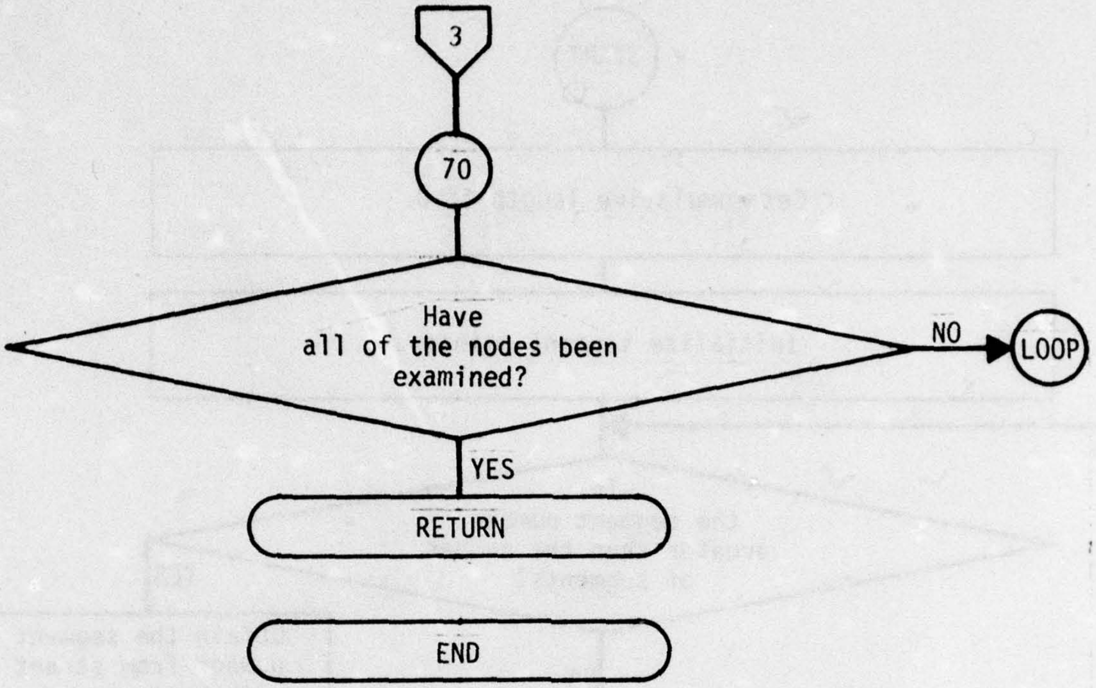
Subroutine ADJUST



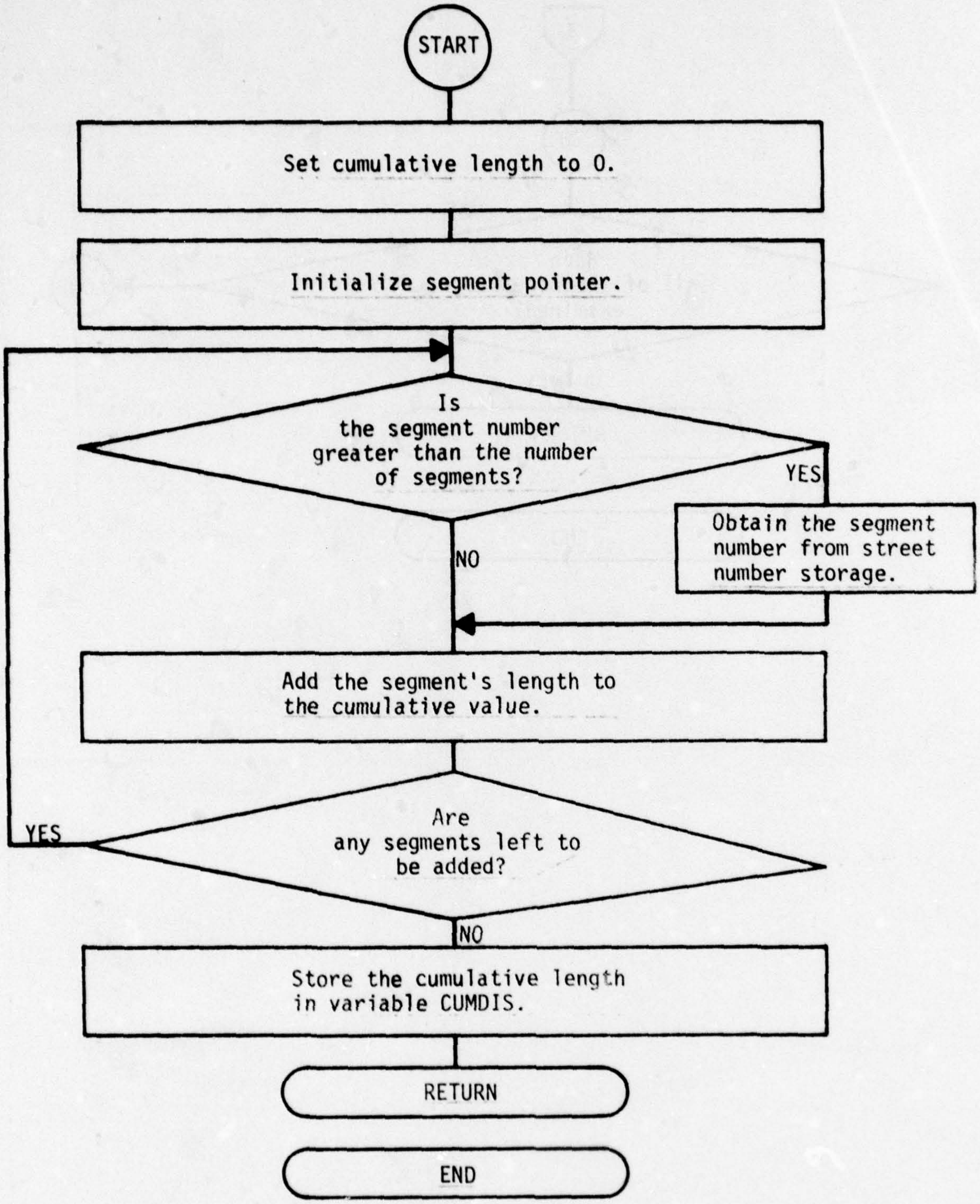
Subroutine ADJUST



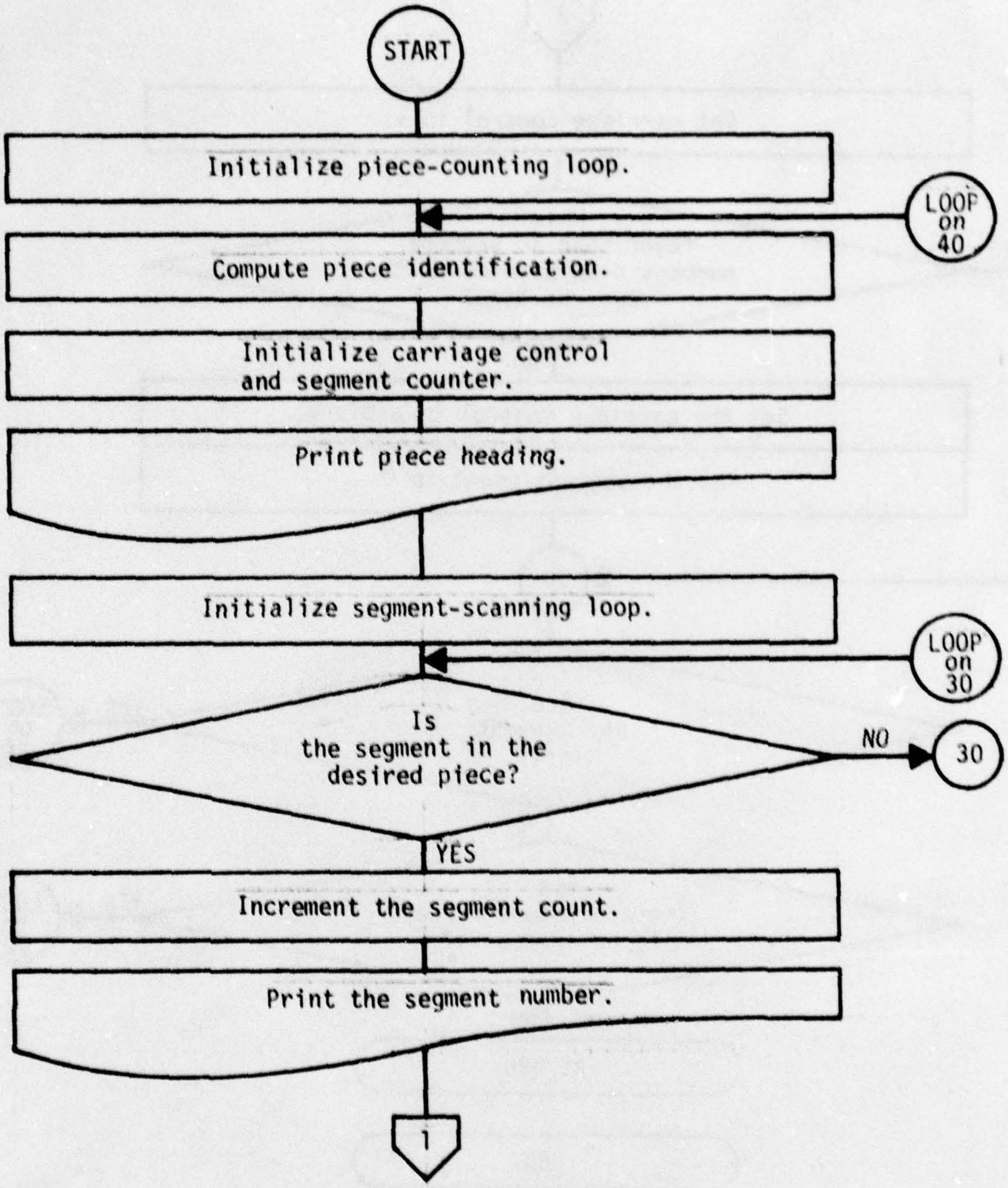
Subroutine ADJUST



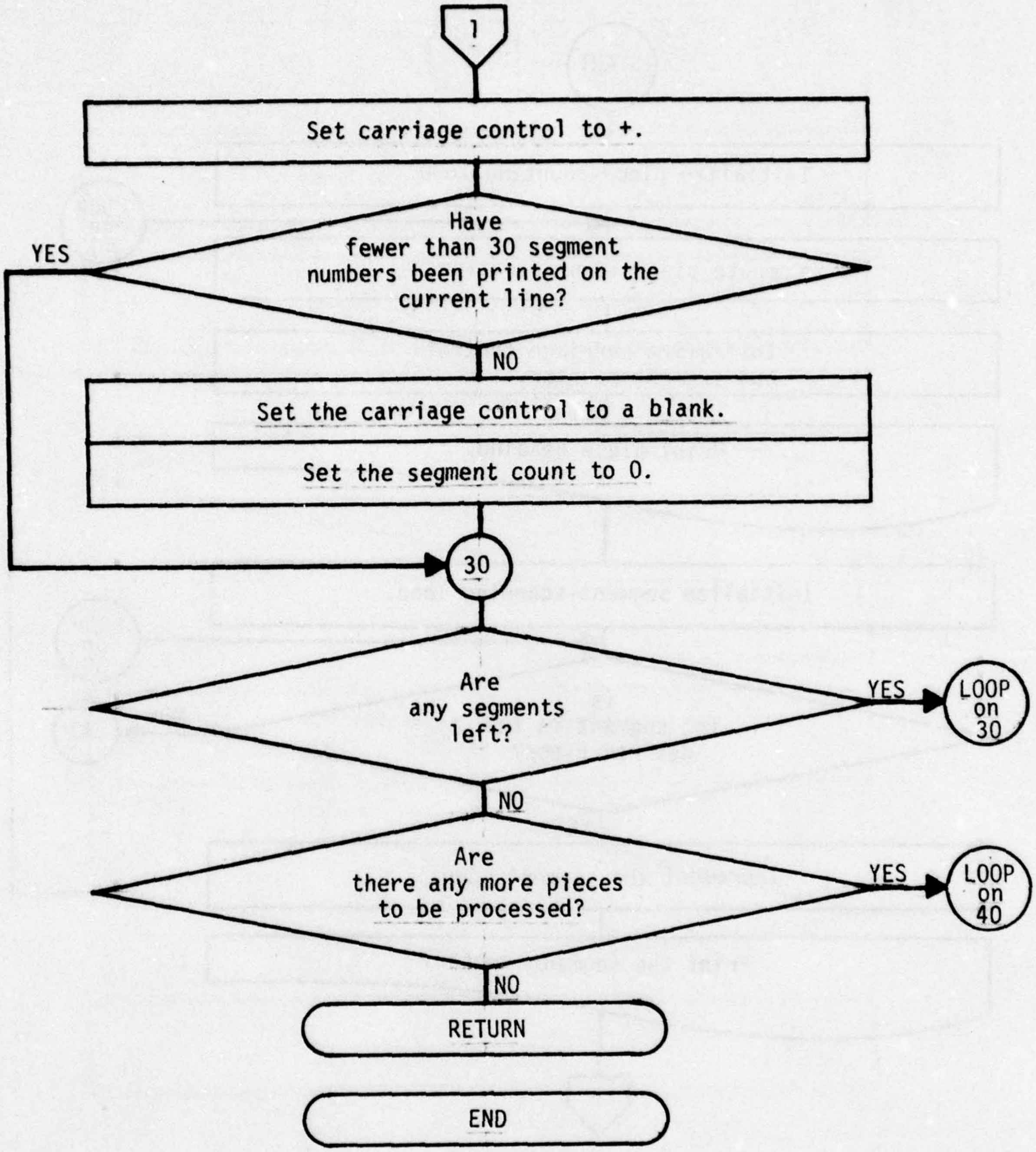
Subroutine ADJUST



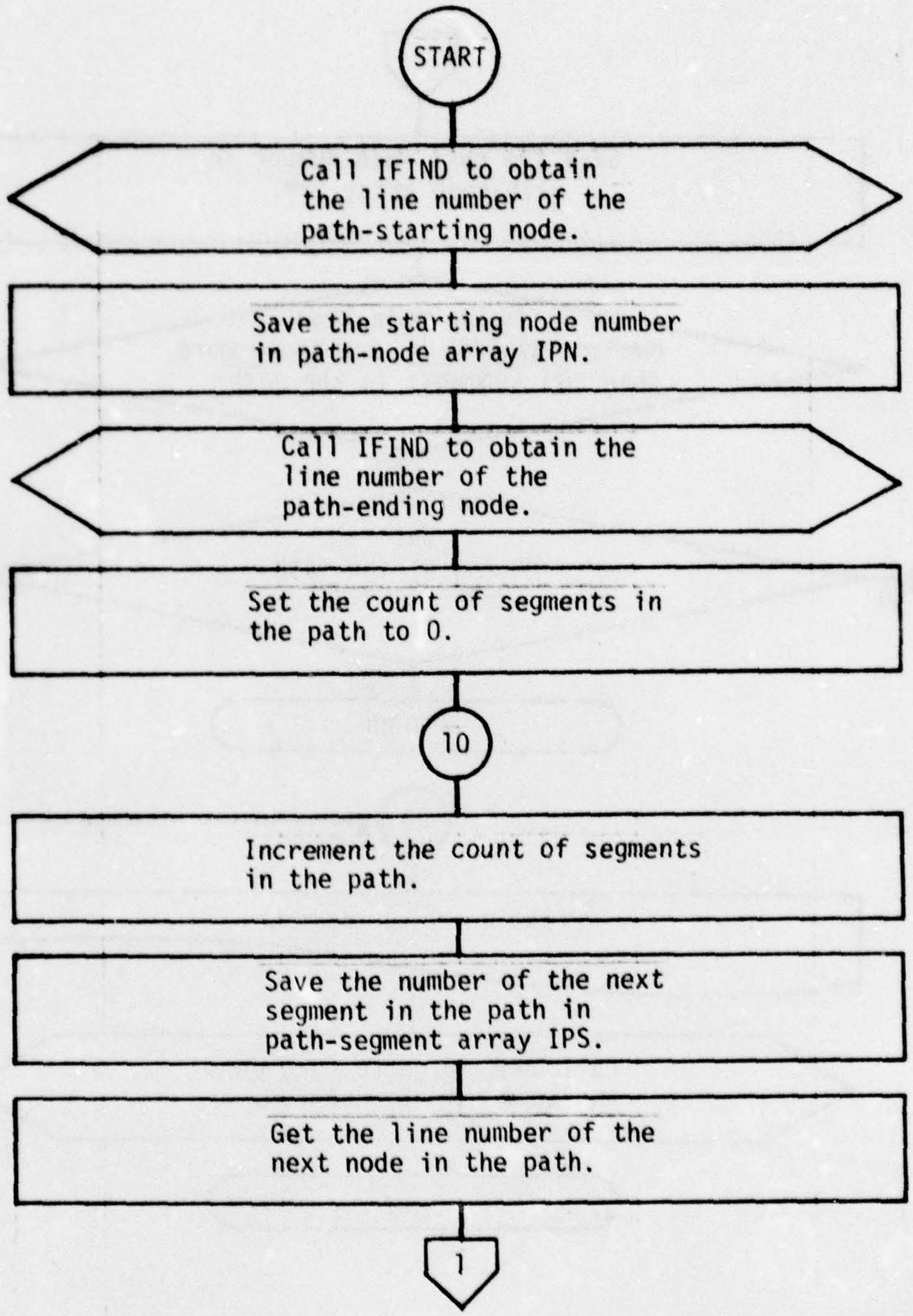
Function CUMDIS



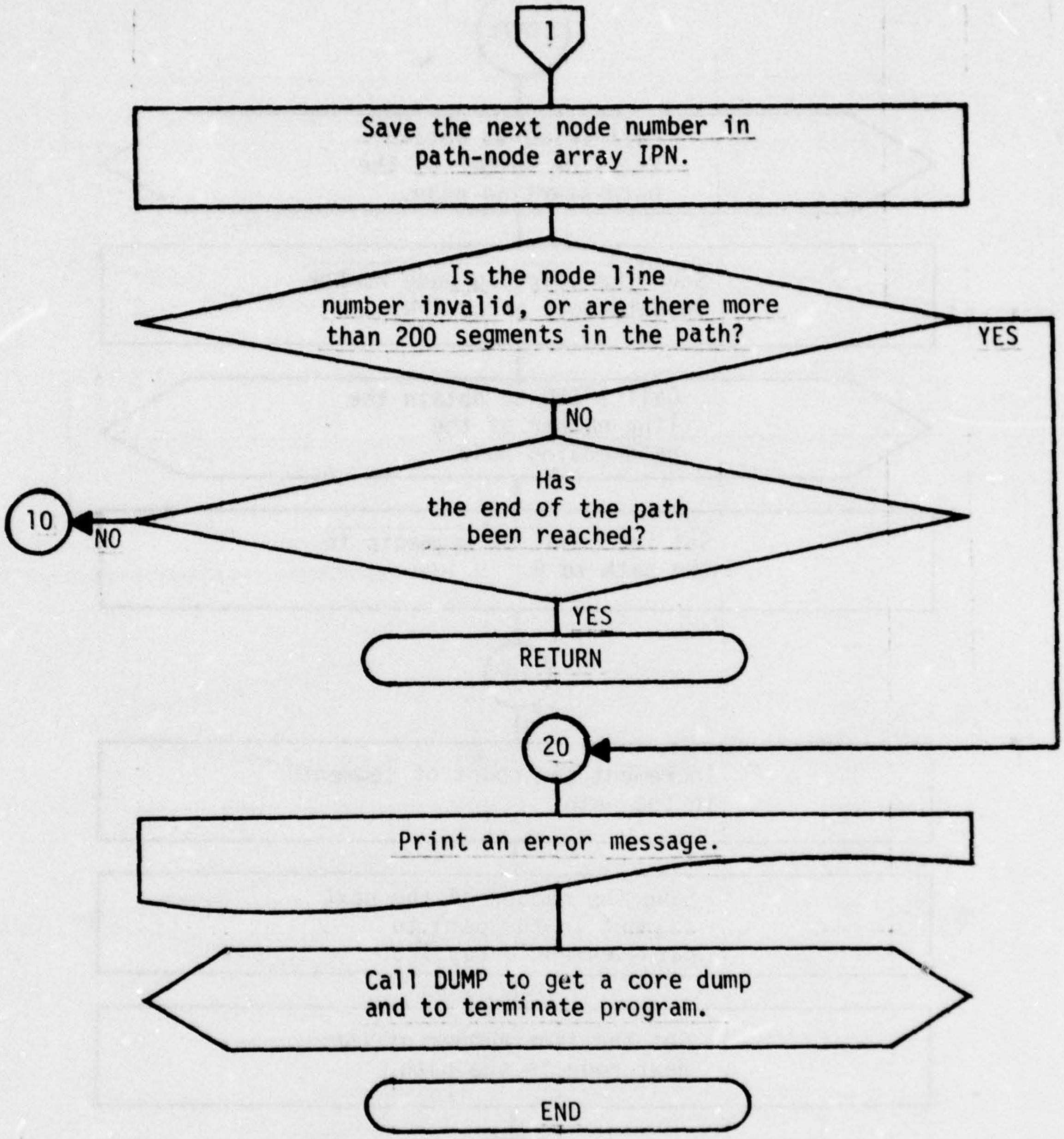
Subroutine PRNPCS



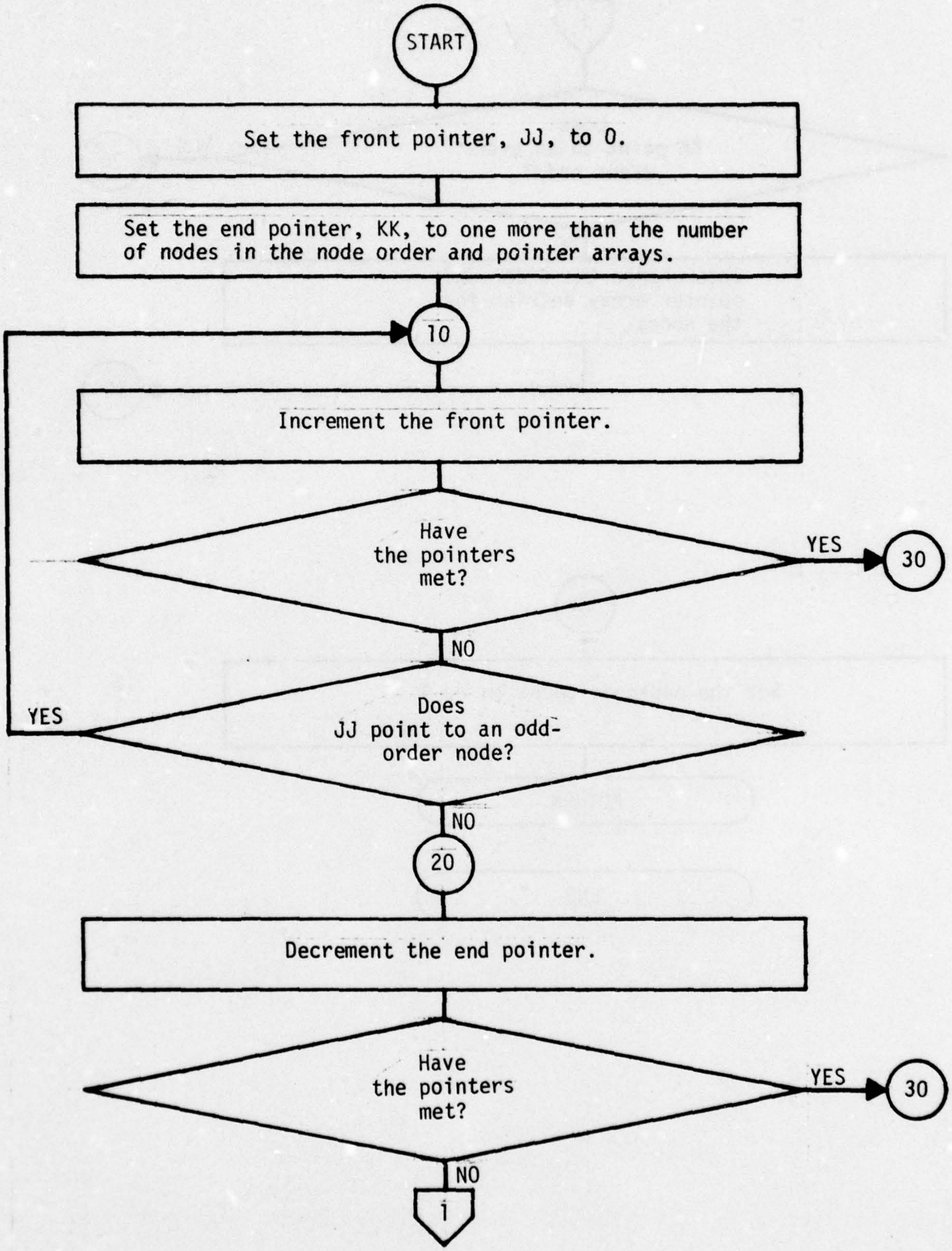
Subroutine PRNPCS



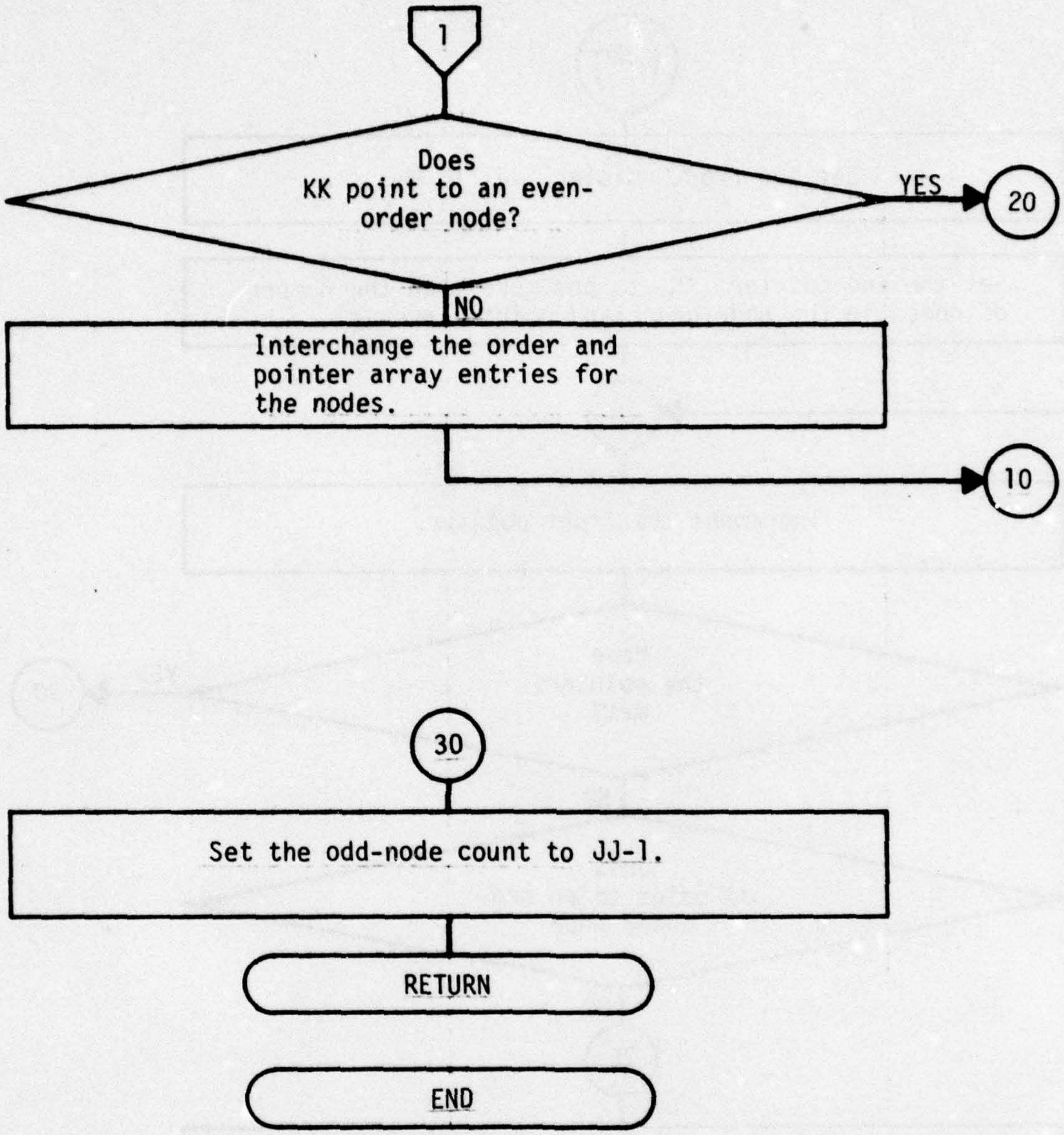
Subroutine TRACE



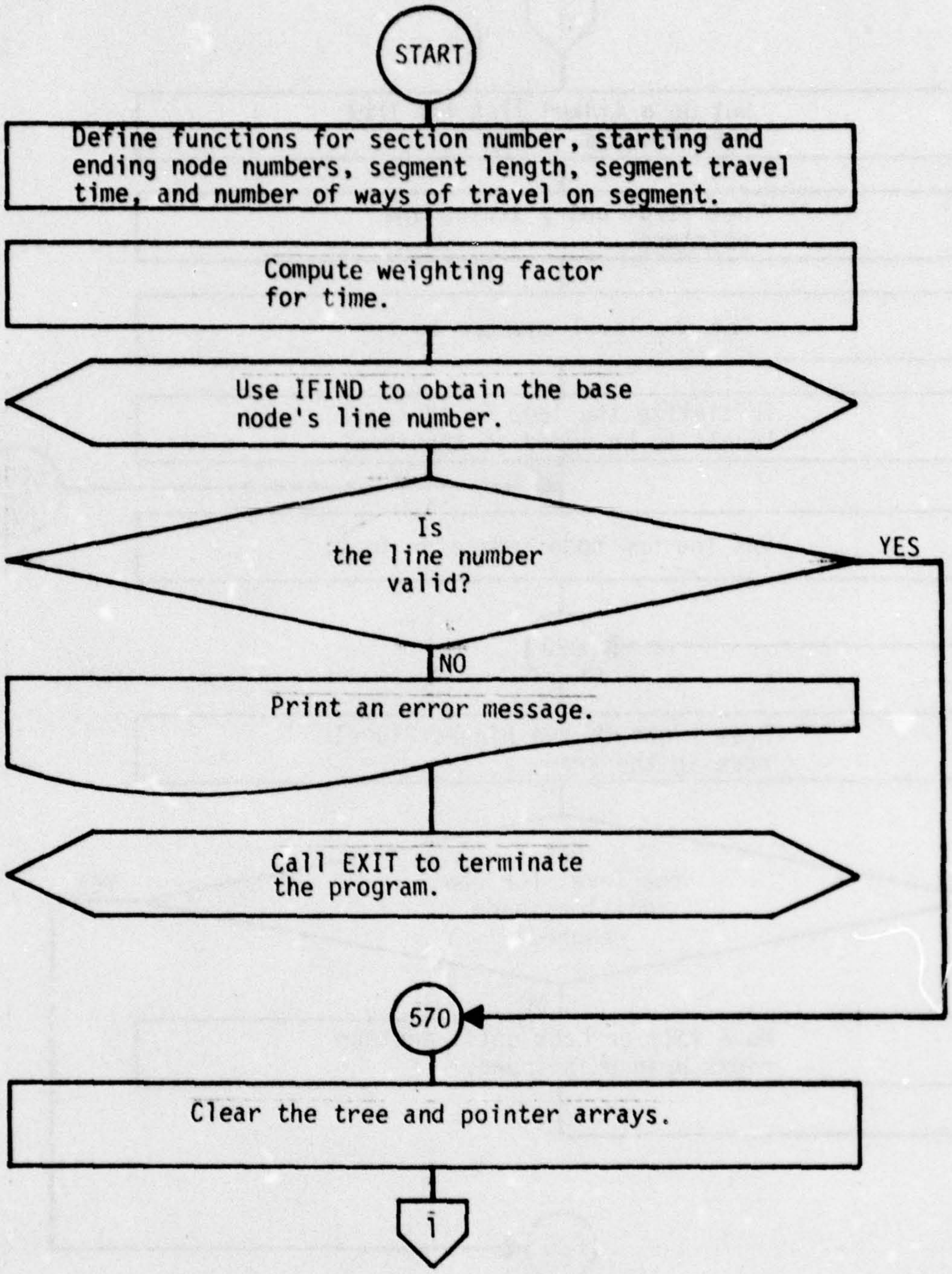
Subroutine TRACE



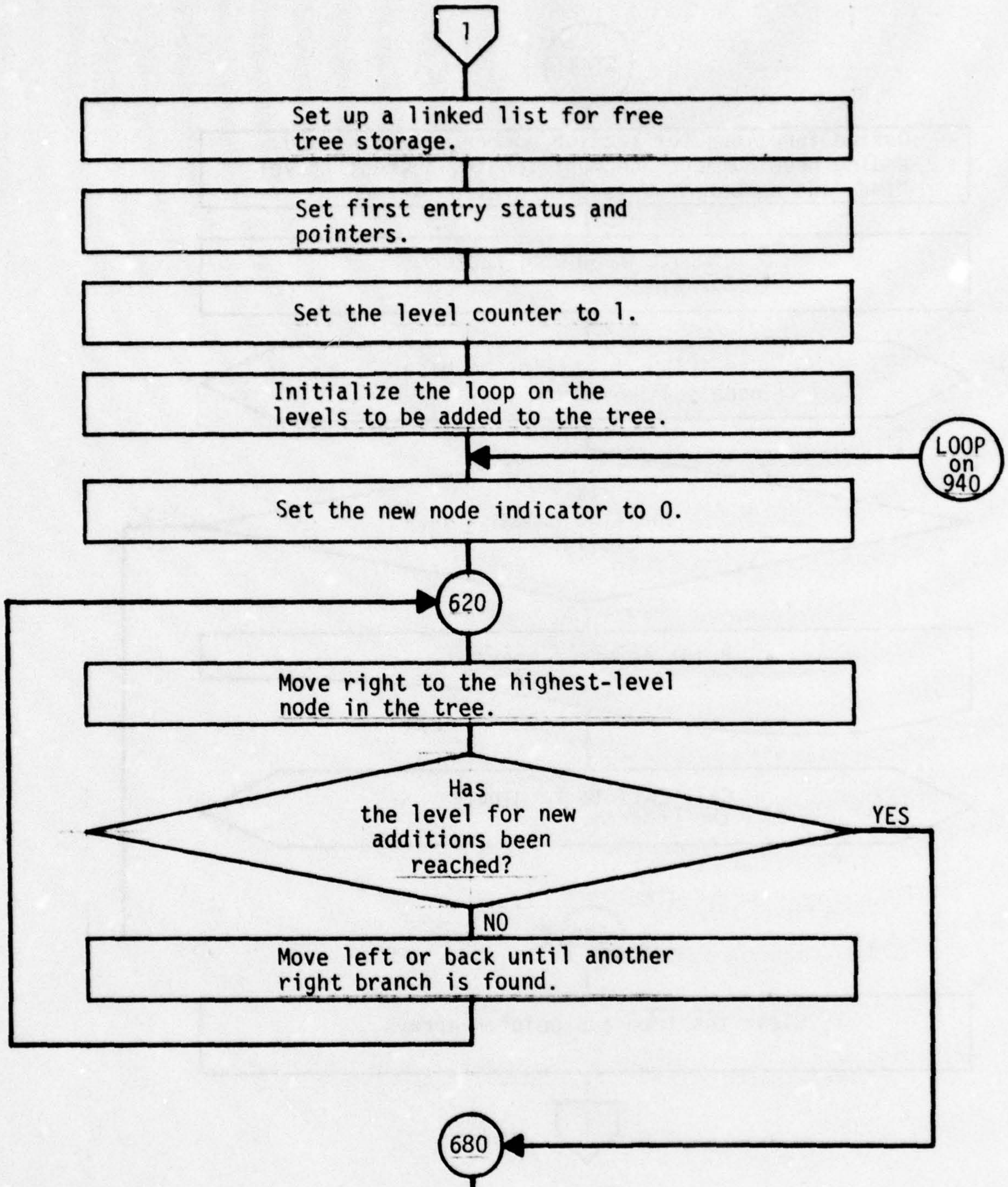
Subroutine MOVODD



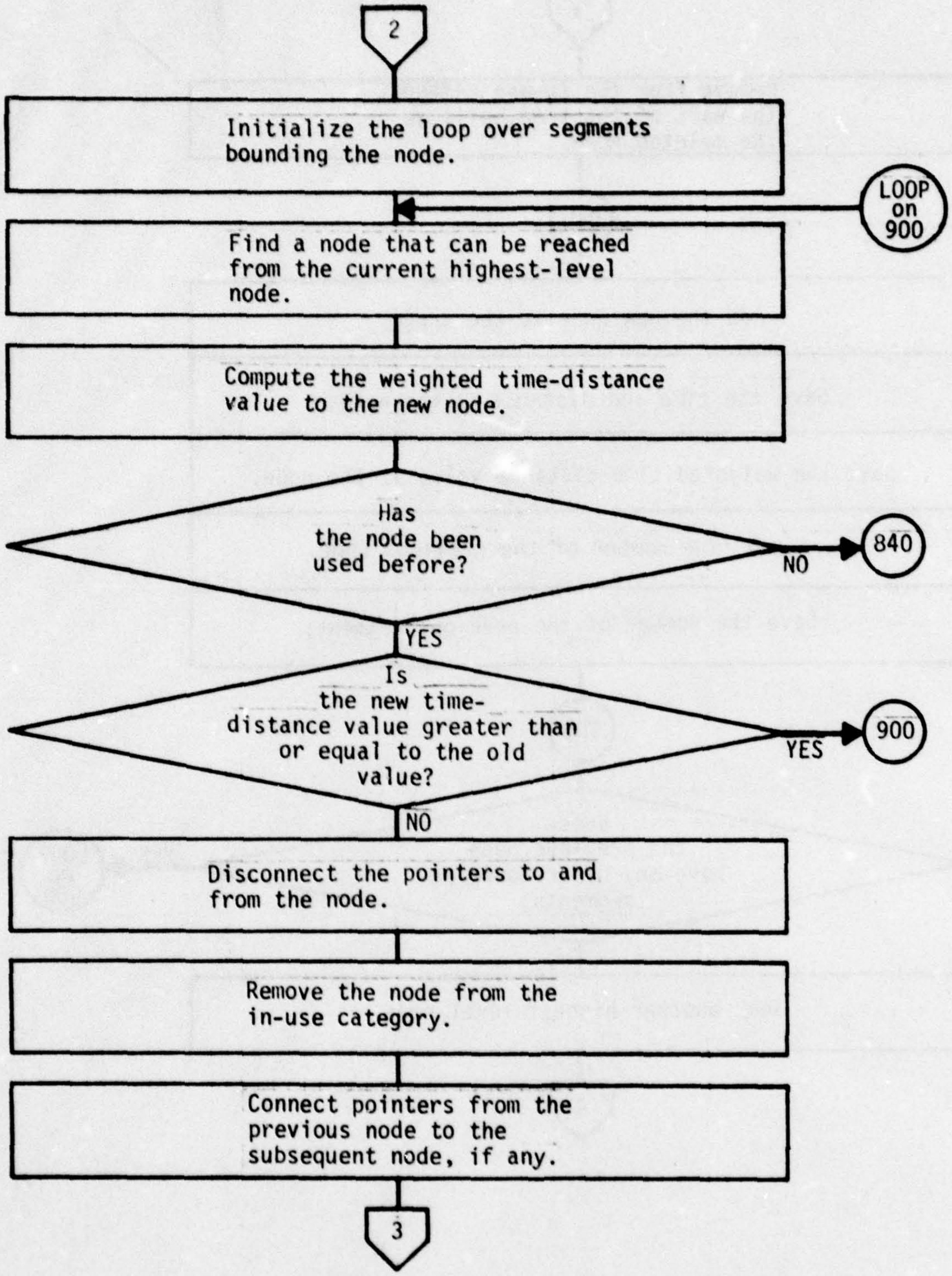
Subroutine MOVODD



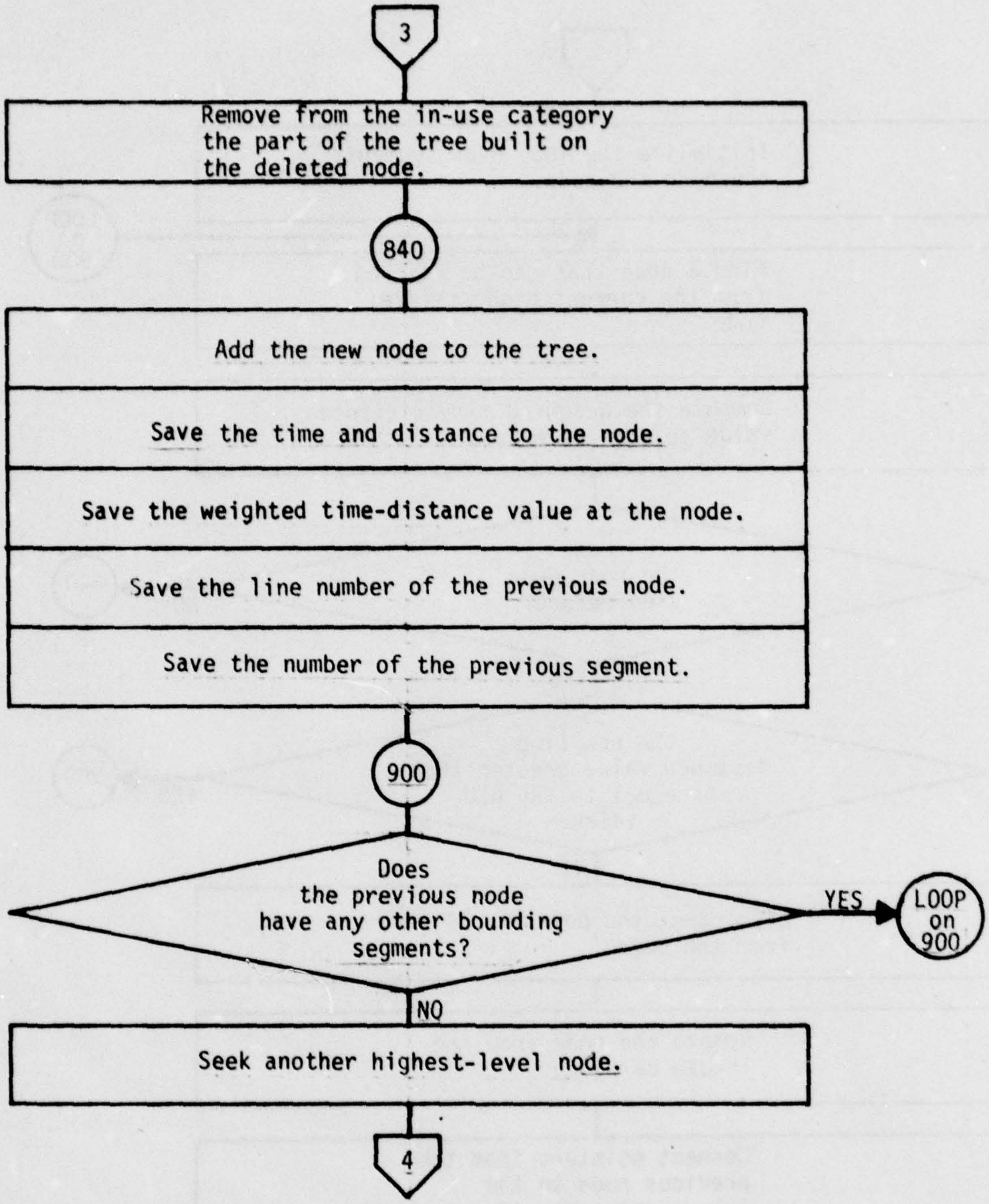
Subroutine TREE



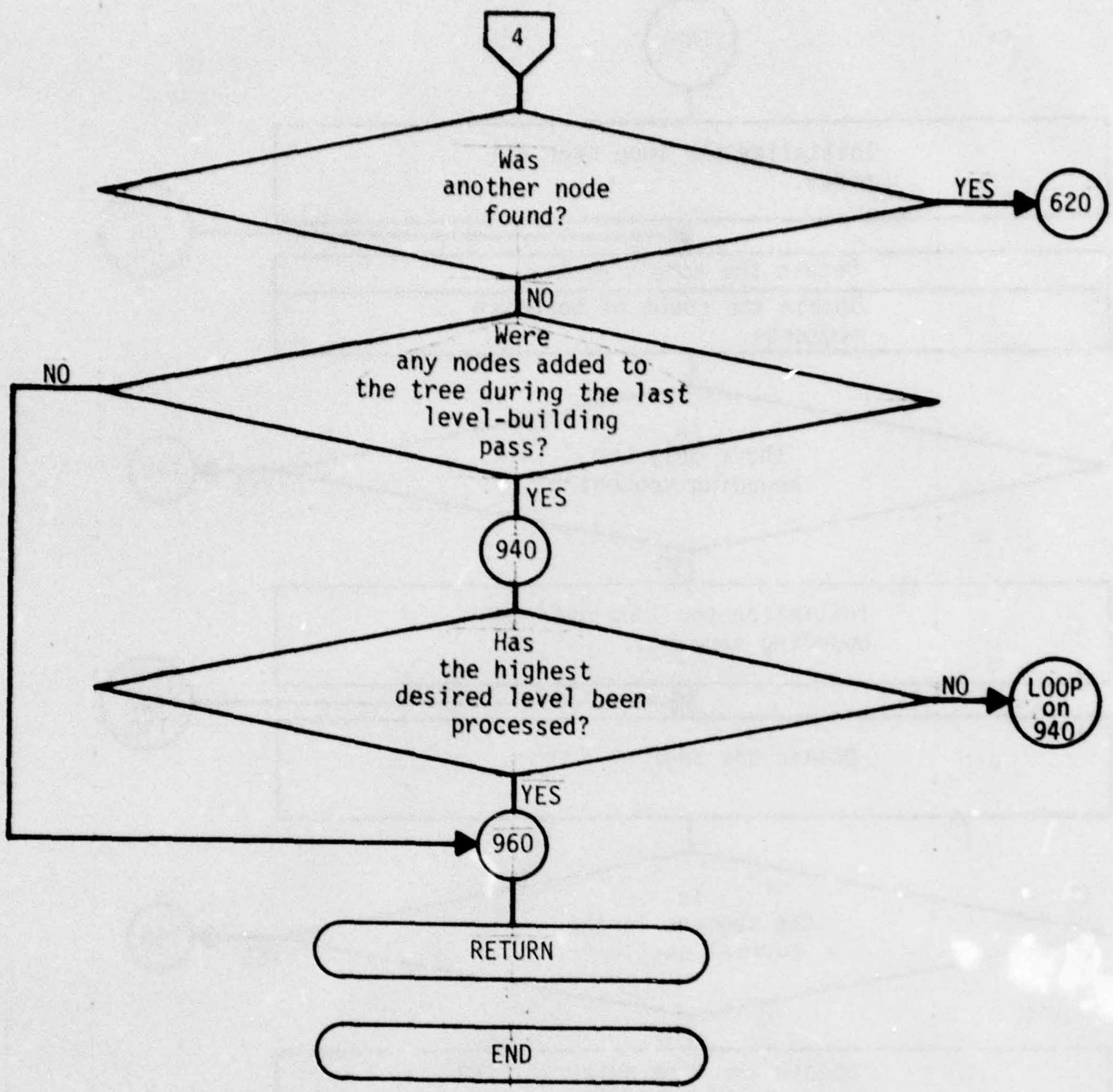
Subroutine TREE



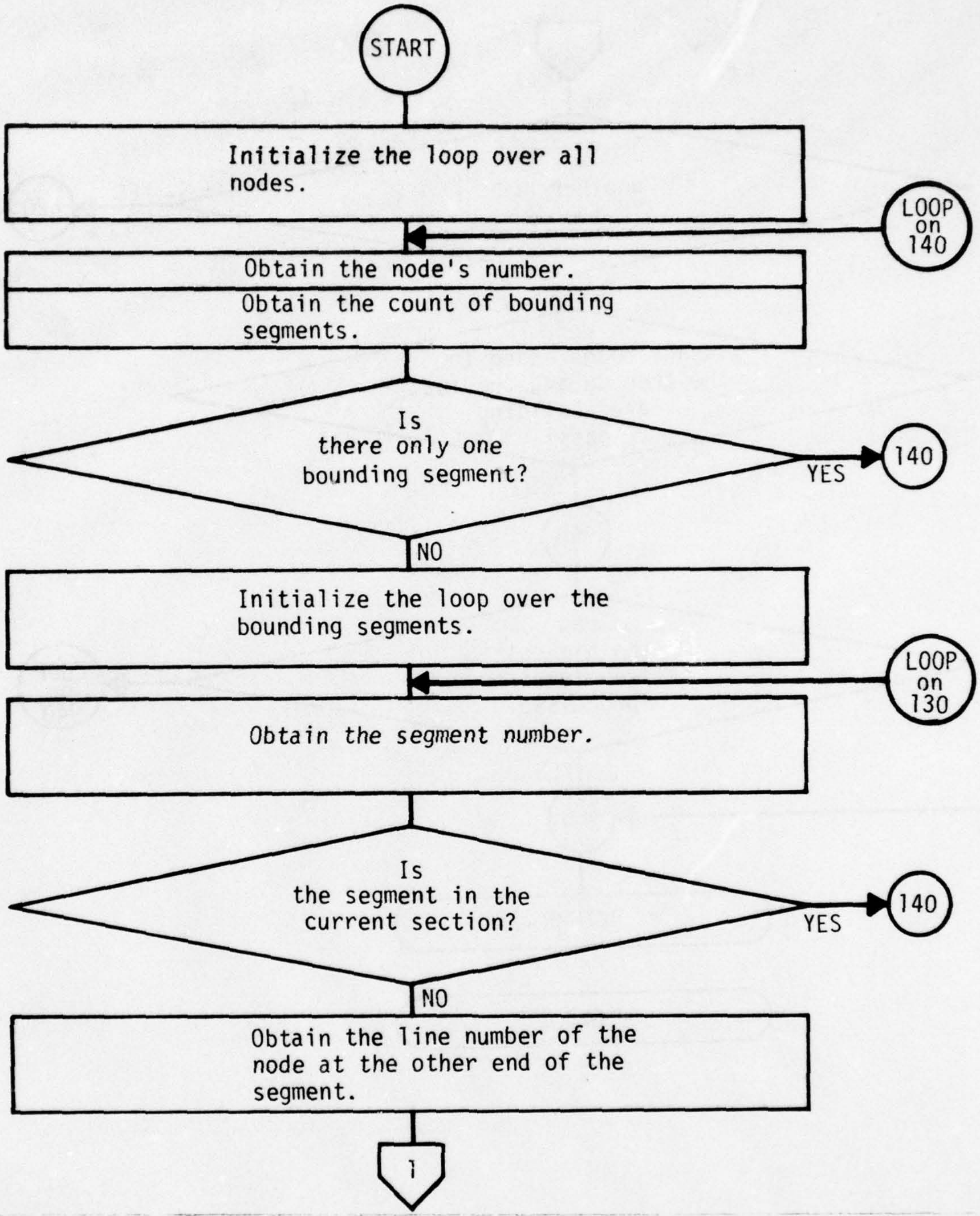
Subroutine TREE



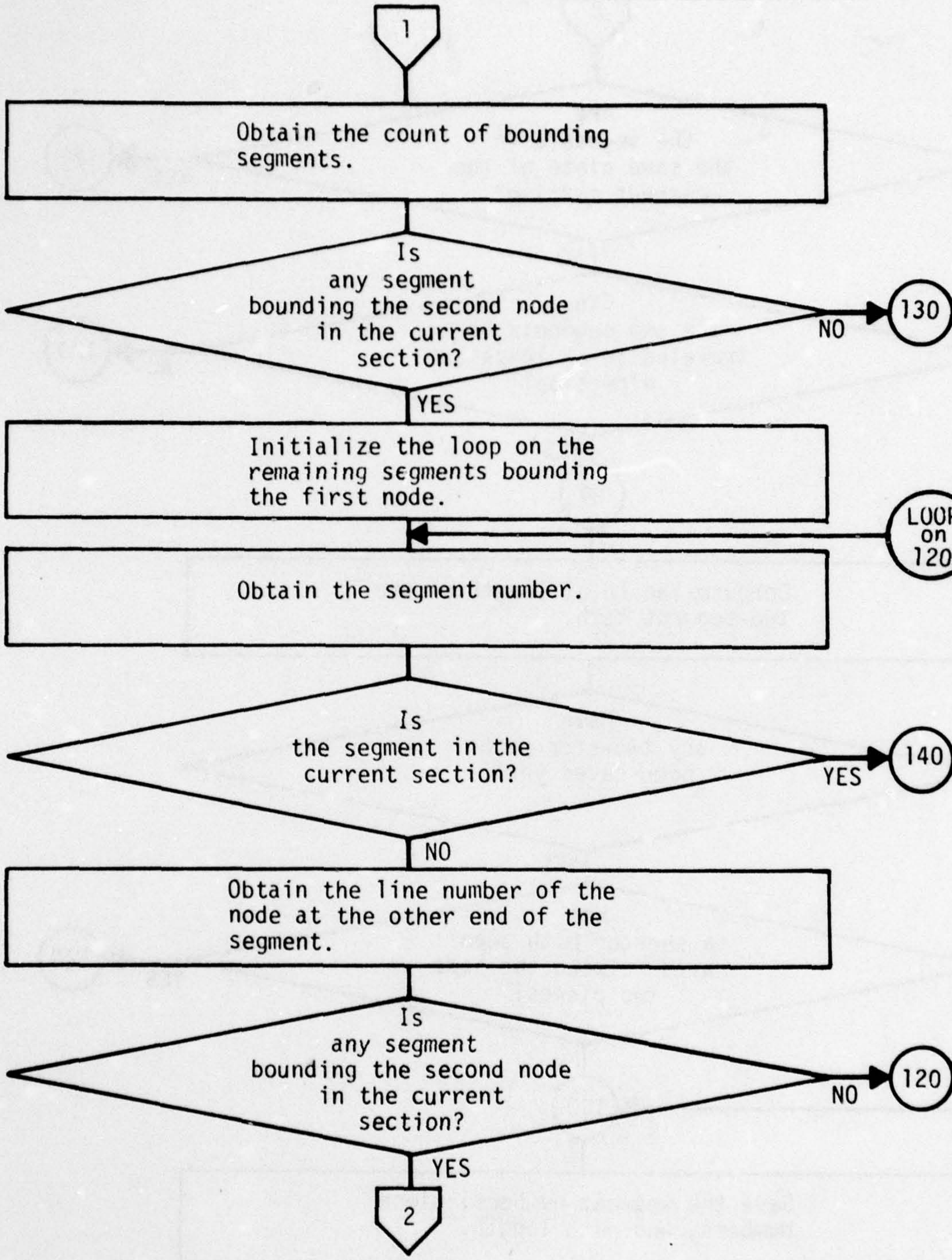
Subroutine TREE



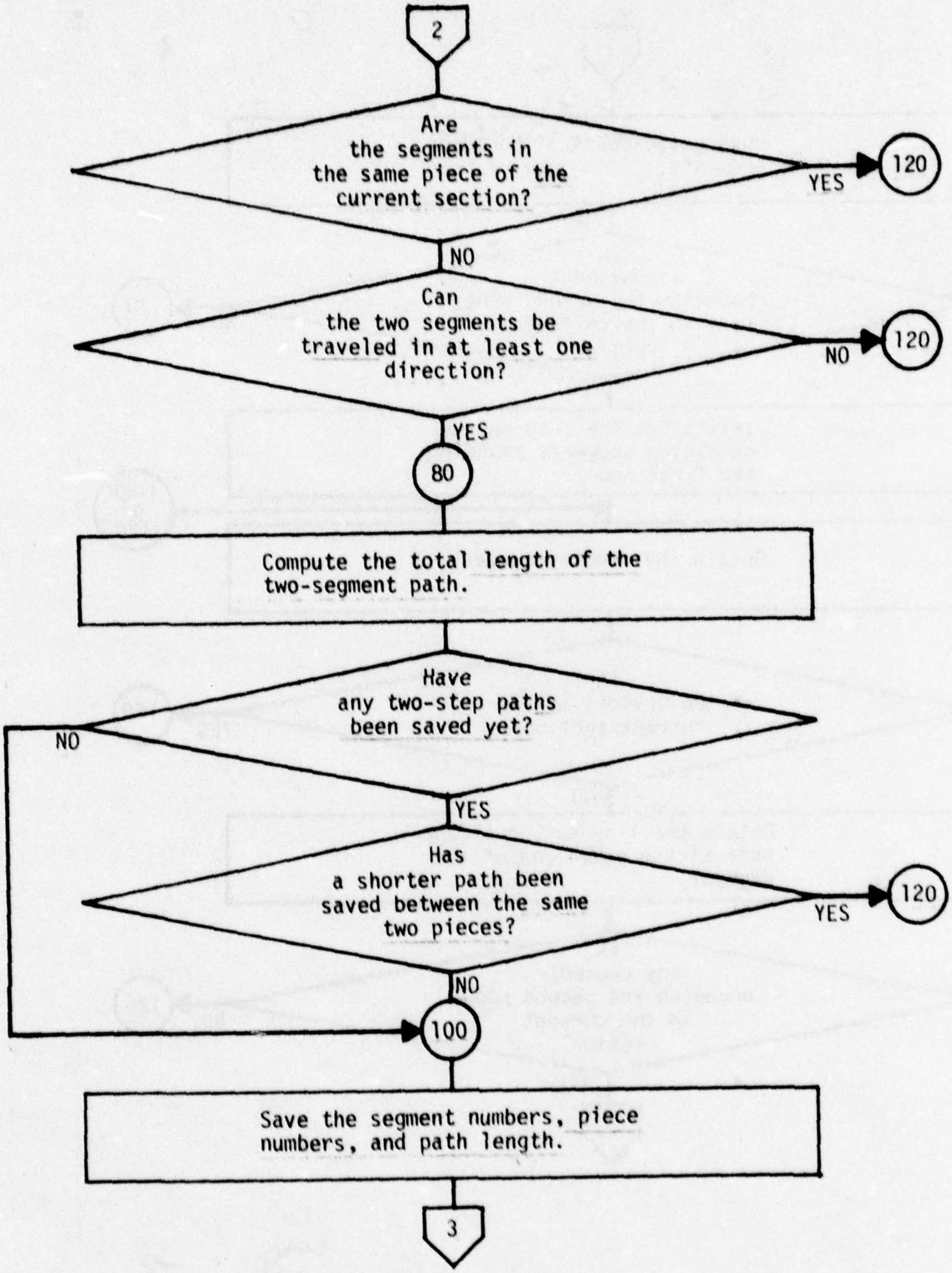
Subroutine TREE



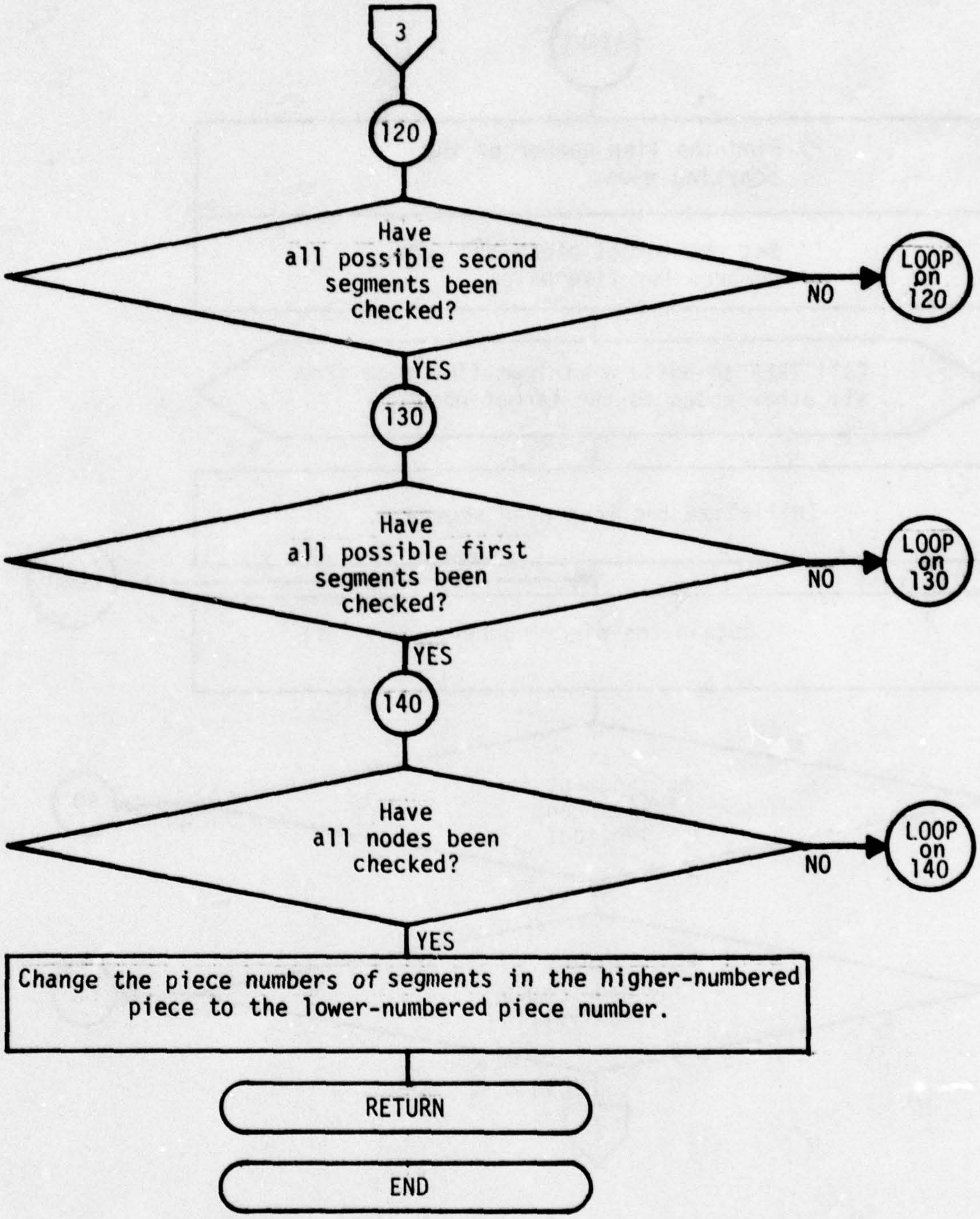
Subroutine CON2ST



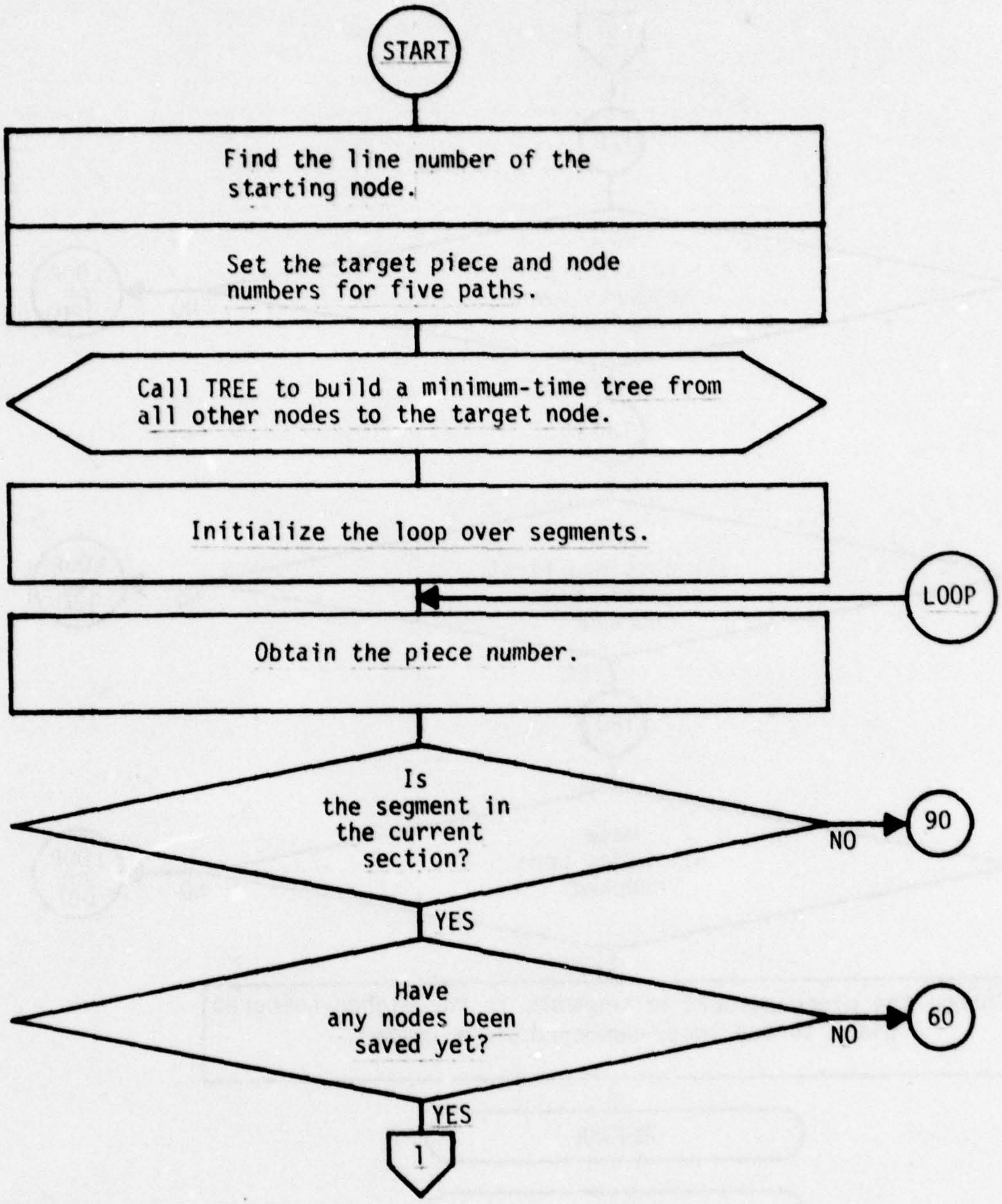
Subroutine CON2ST



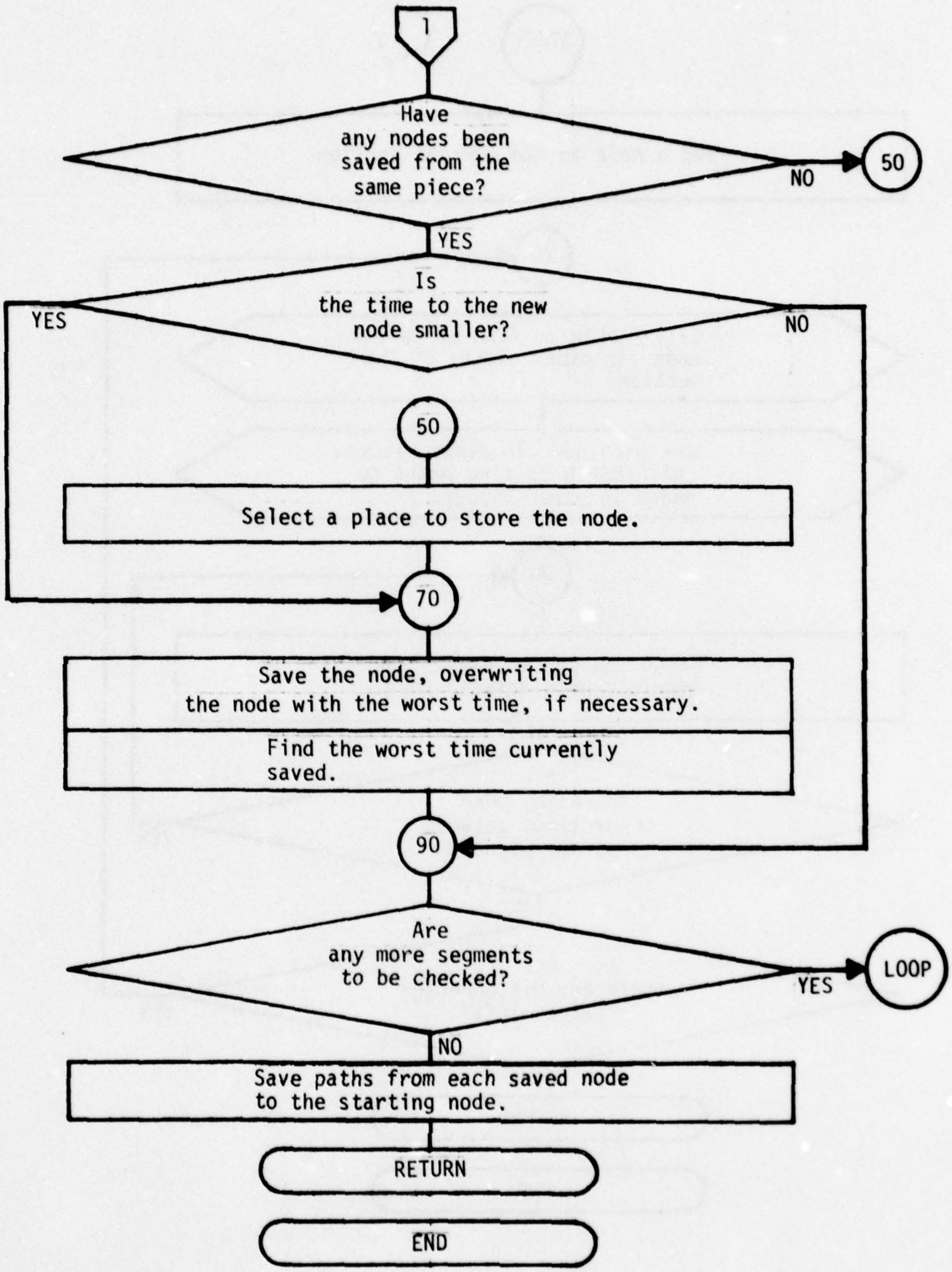
Subroutine CON2ST



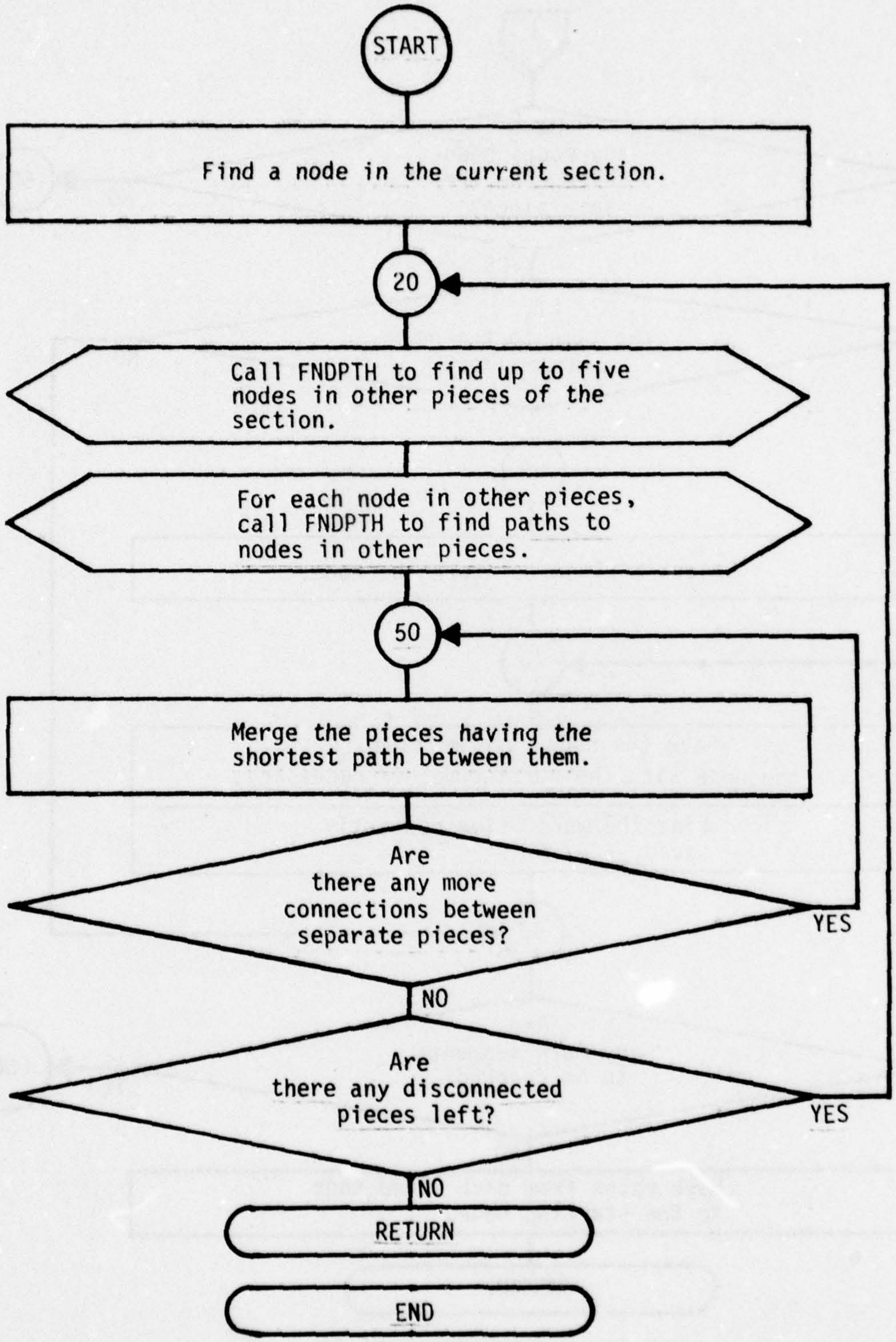
Subroutine CON2ST



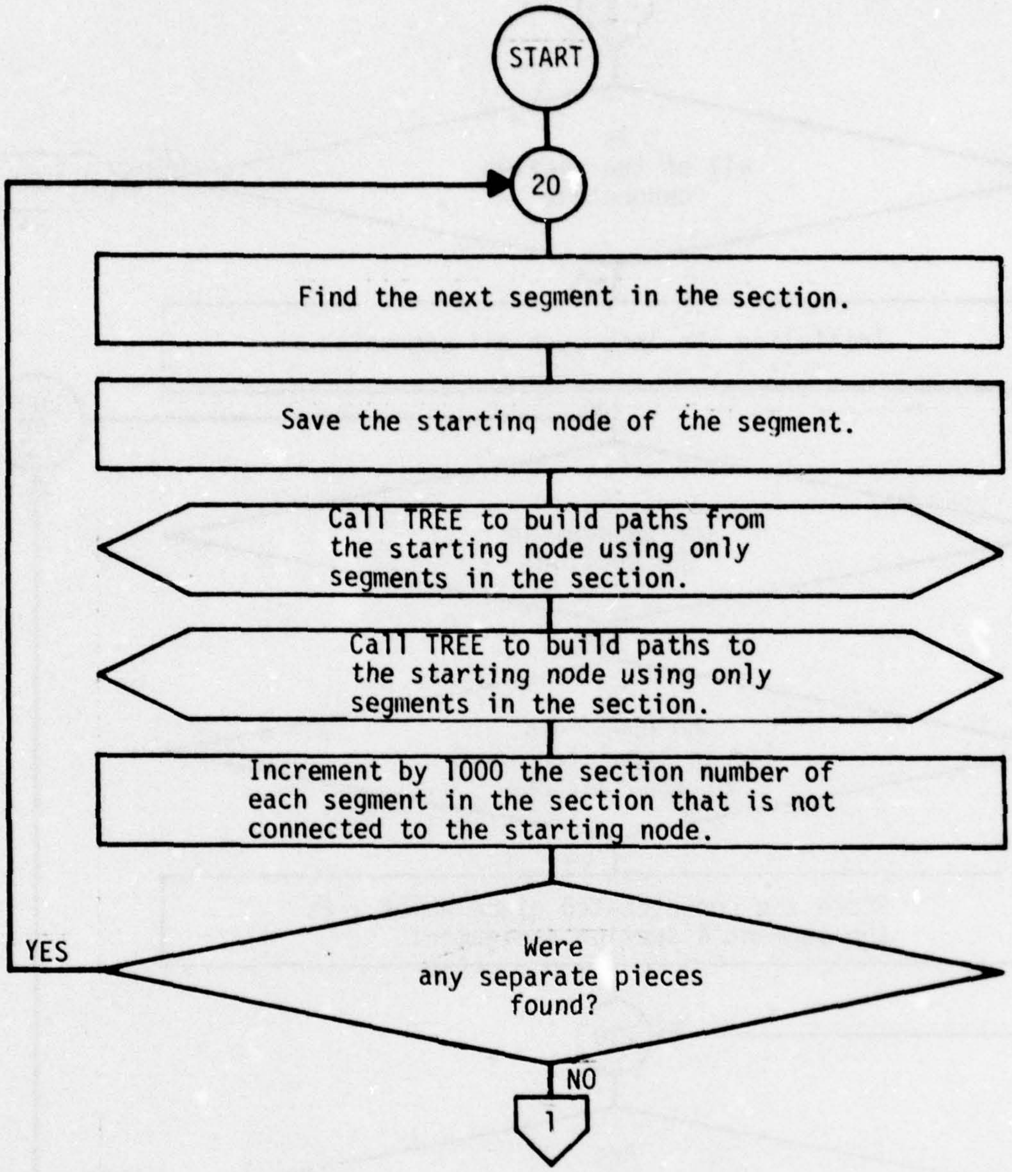
Subroutine FNDPTH



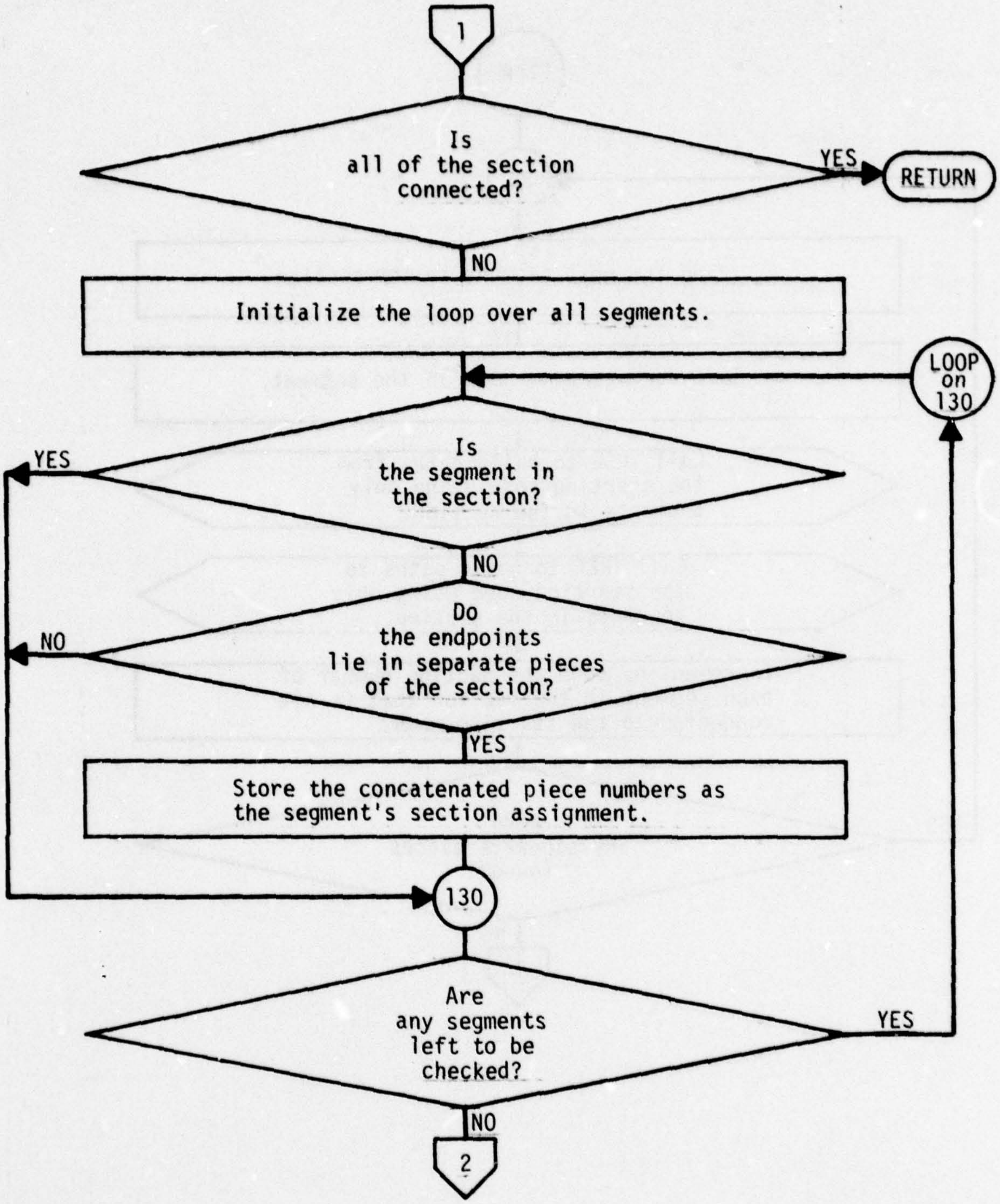
Subroutine FNDPTH



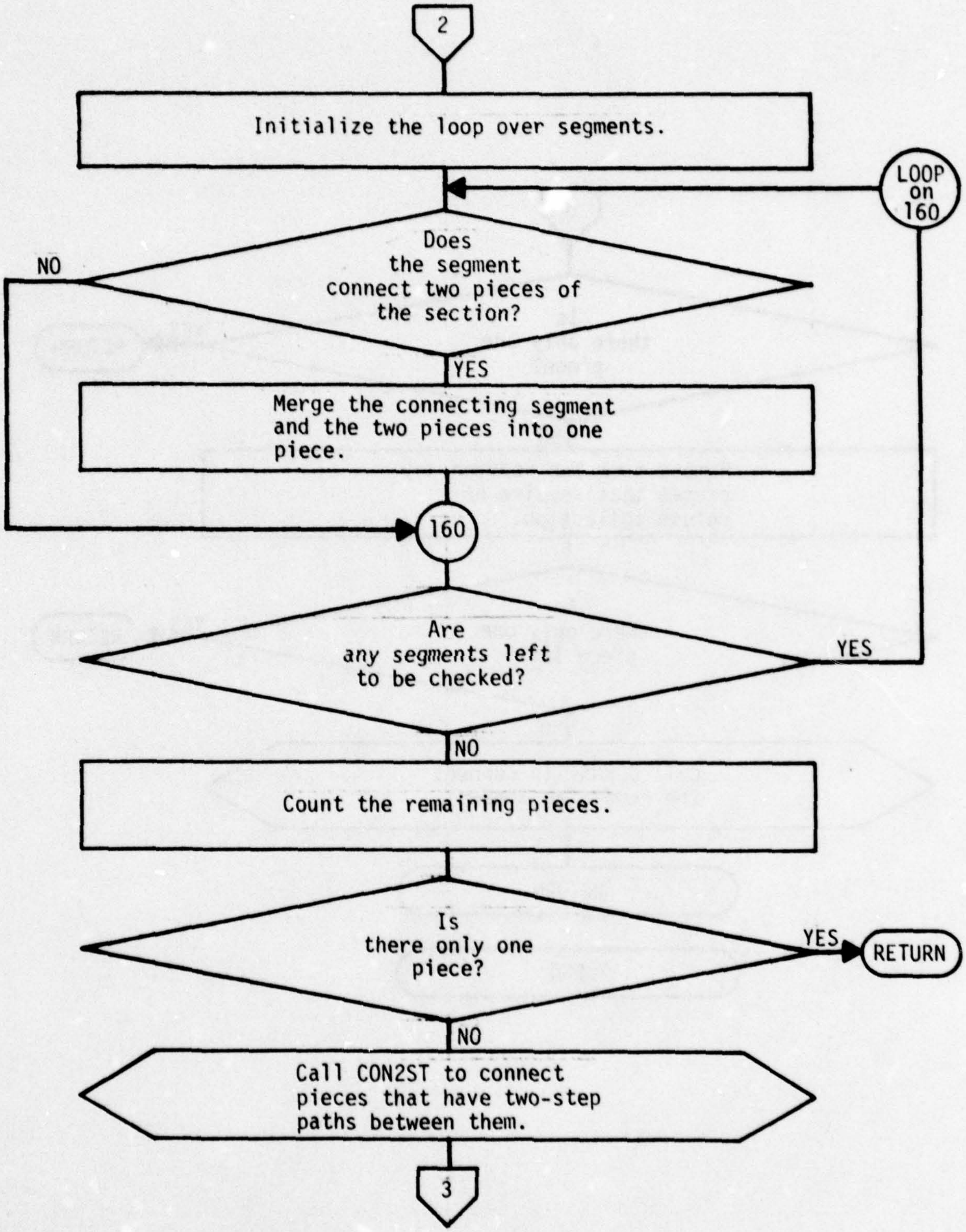
Subroutine CONNST



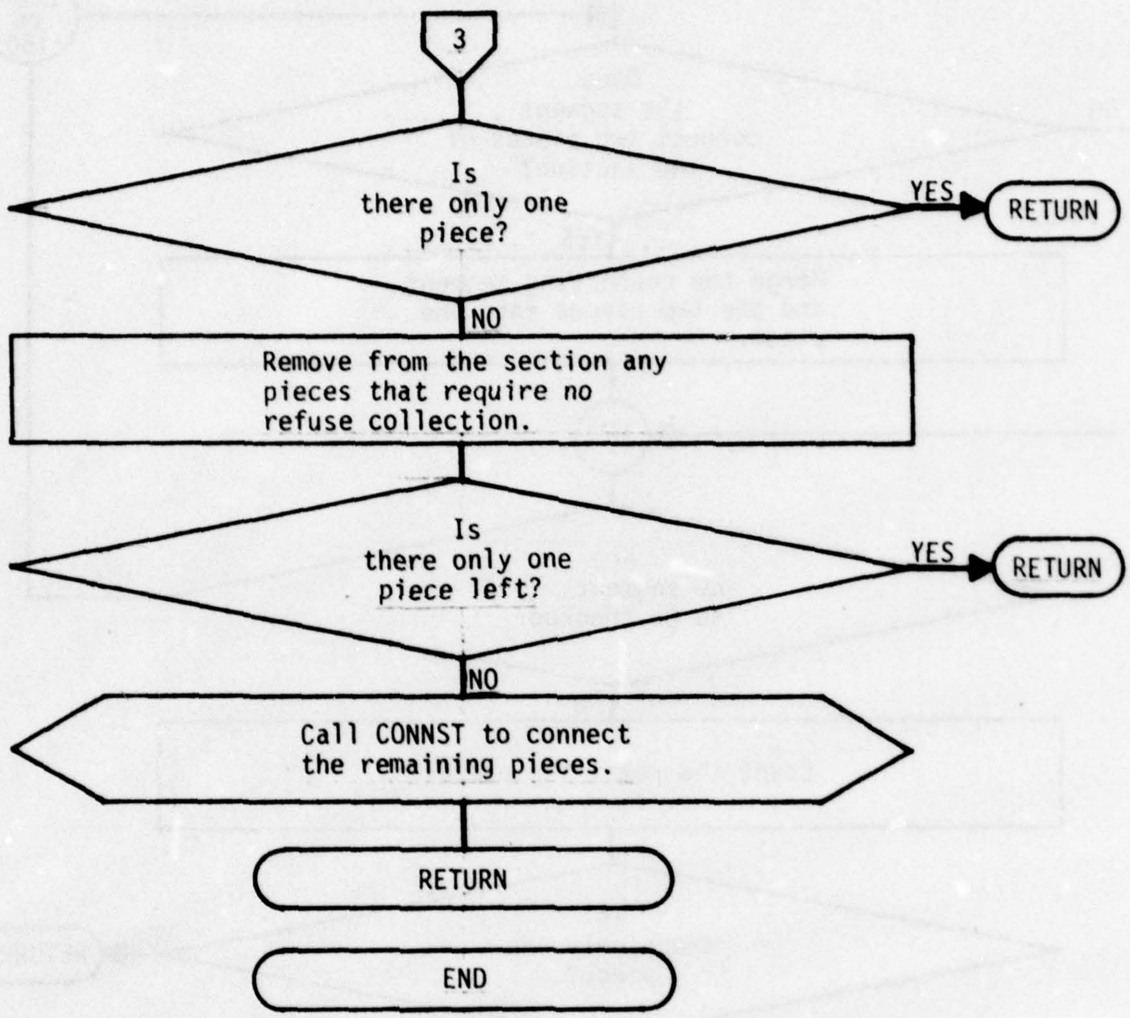
Subroutine CONNECT



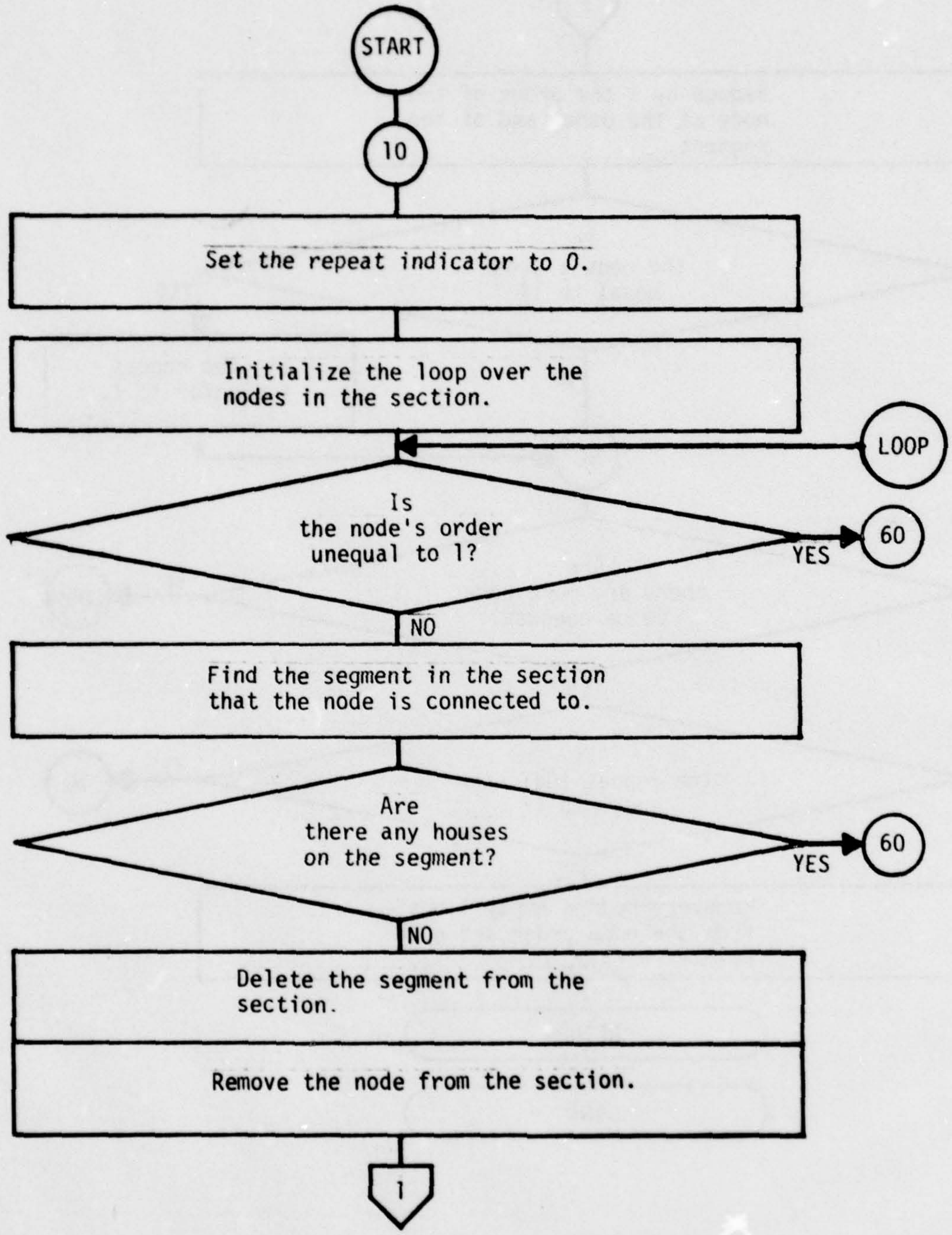
Subroutine CONNECT



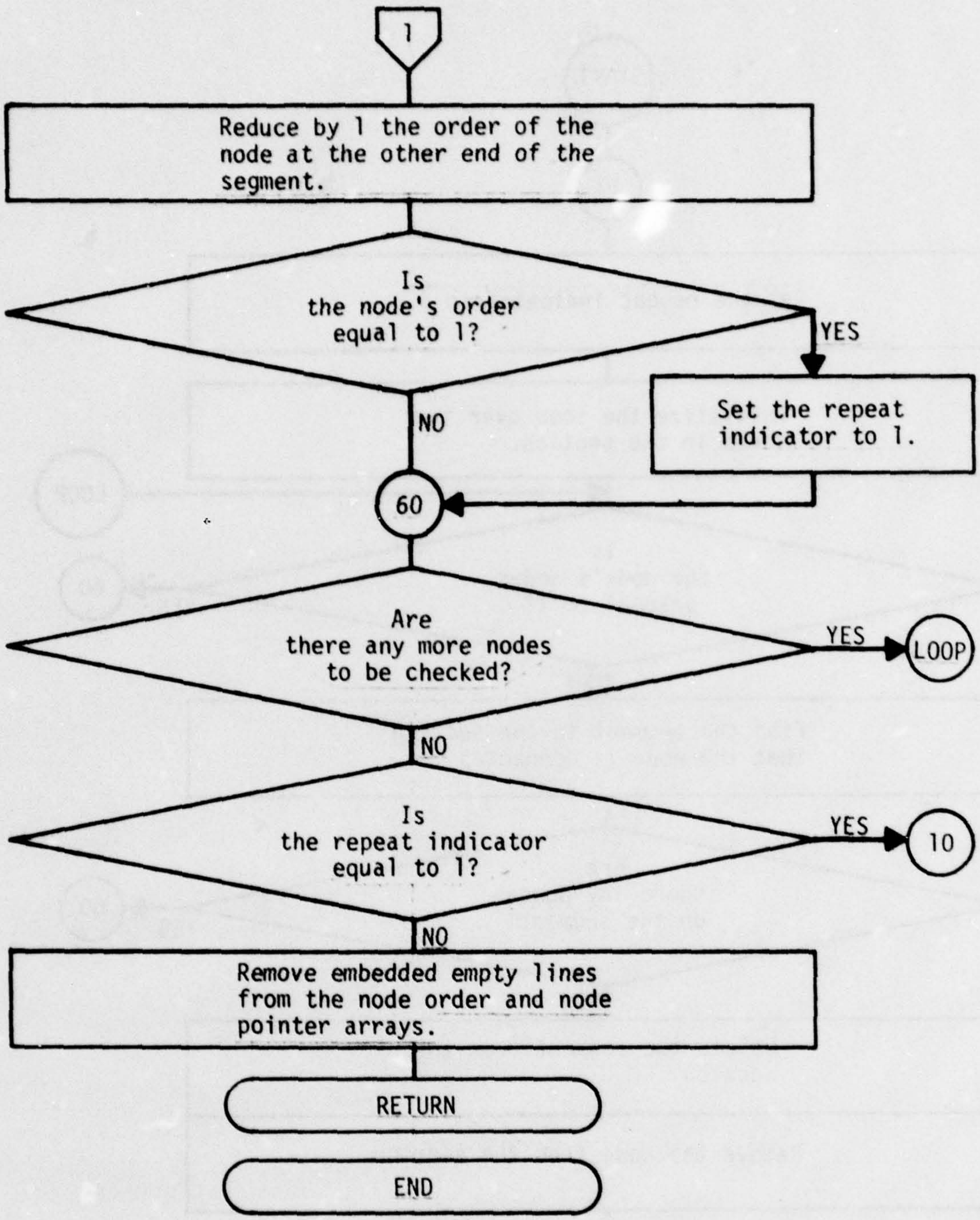
Subroutine CONNECT



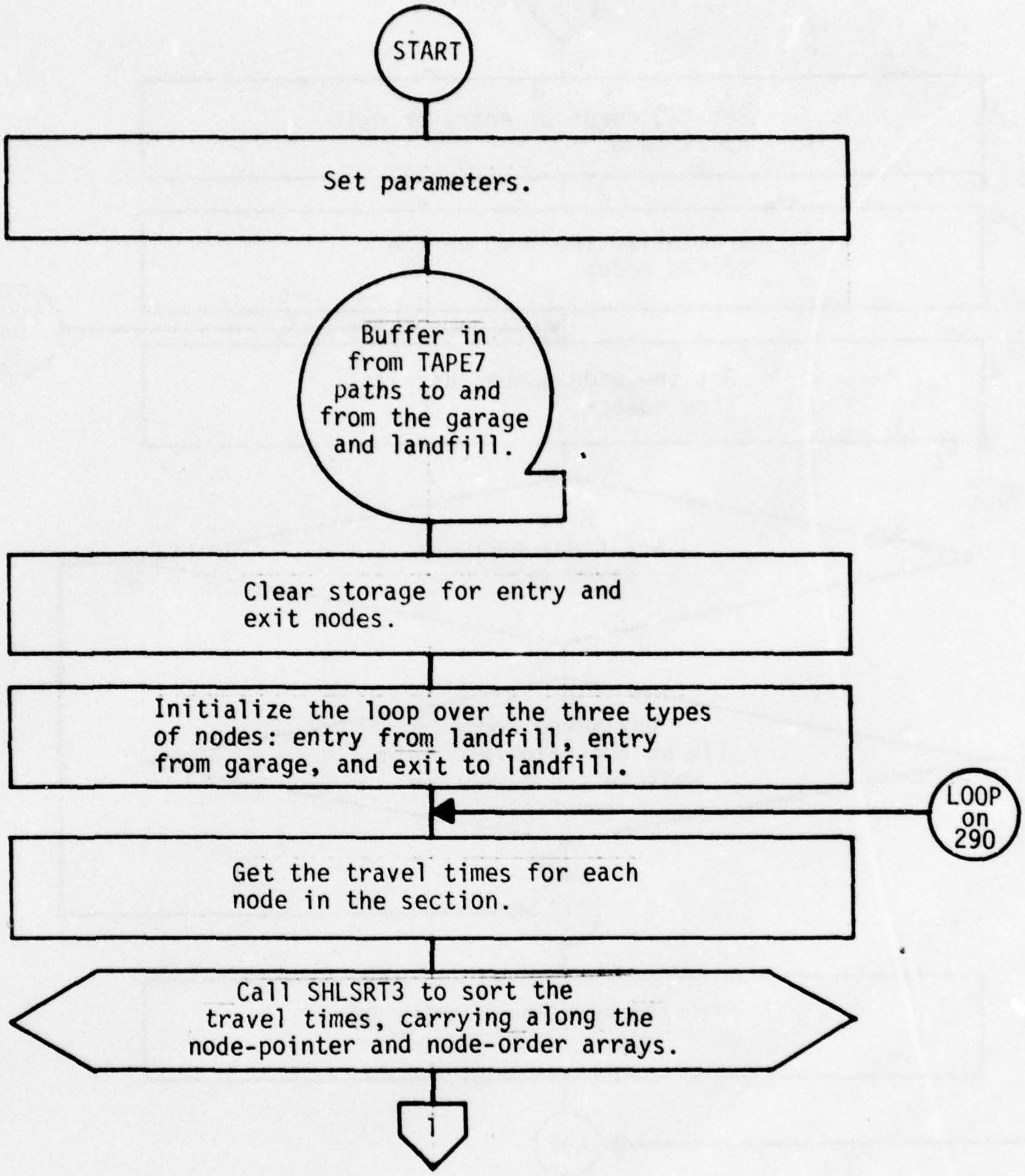
Subroutine CONNECT



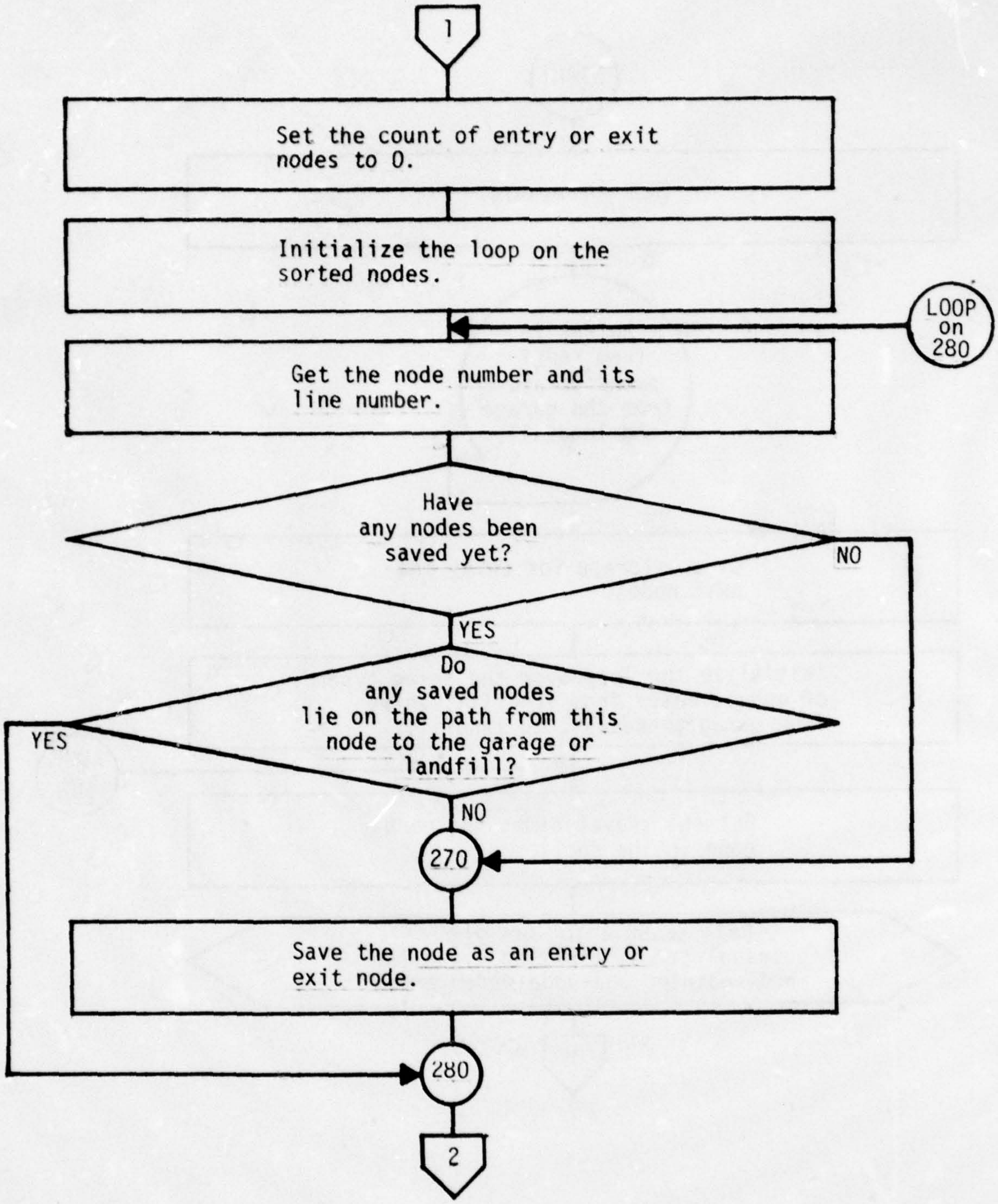
Subroutine DISCON



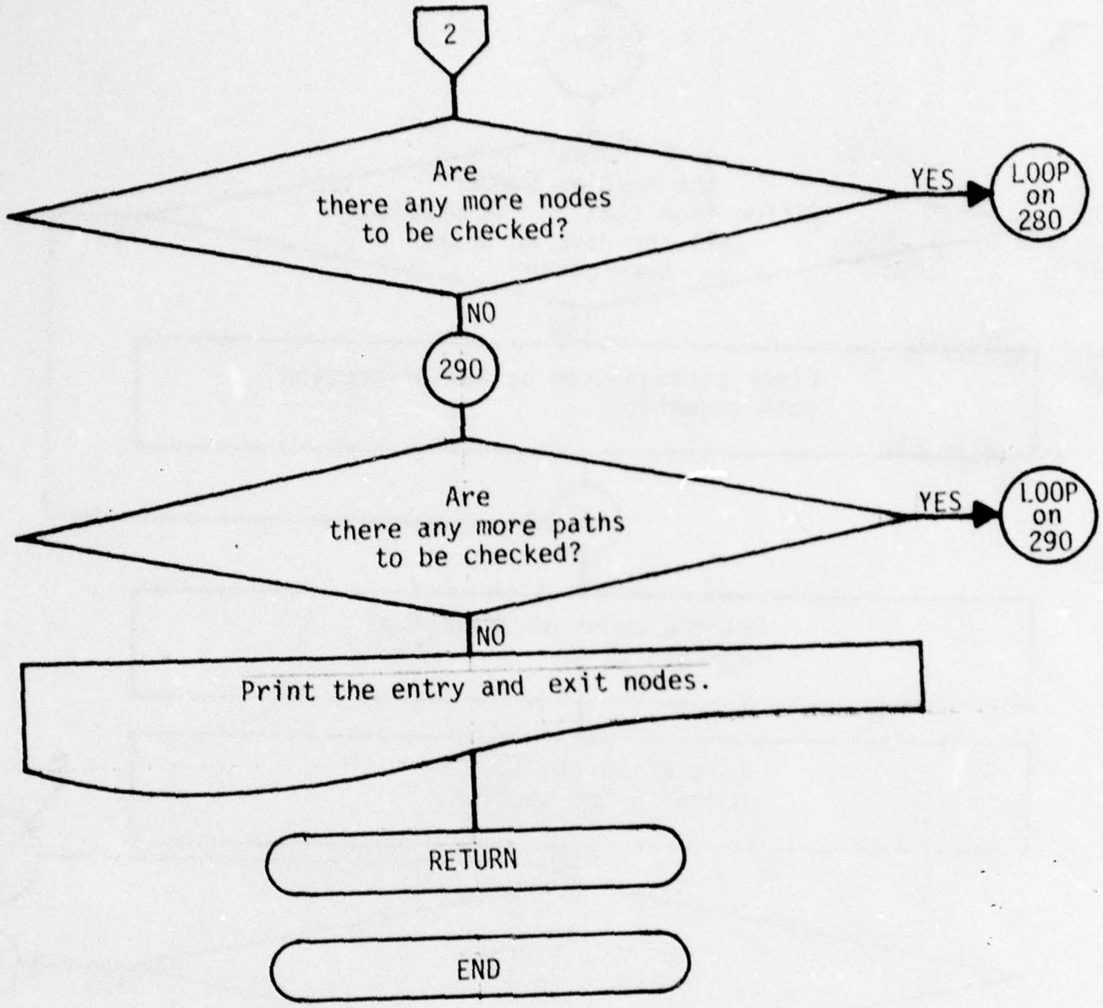
Subroutine DISCON



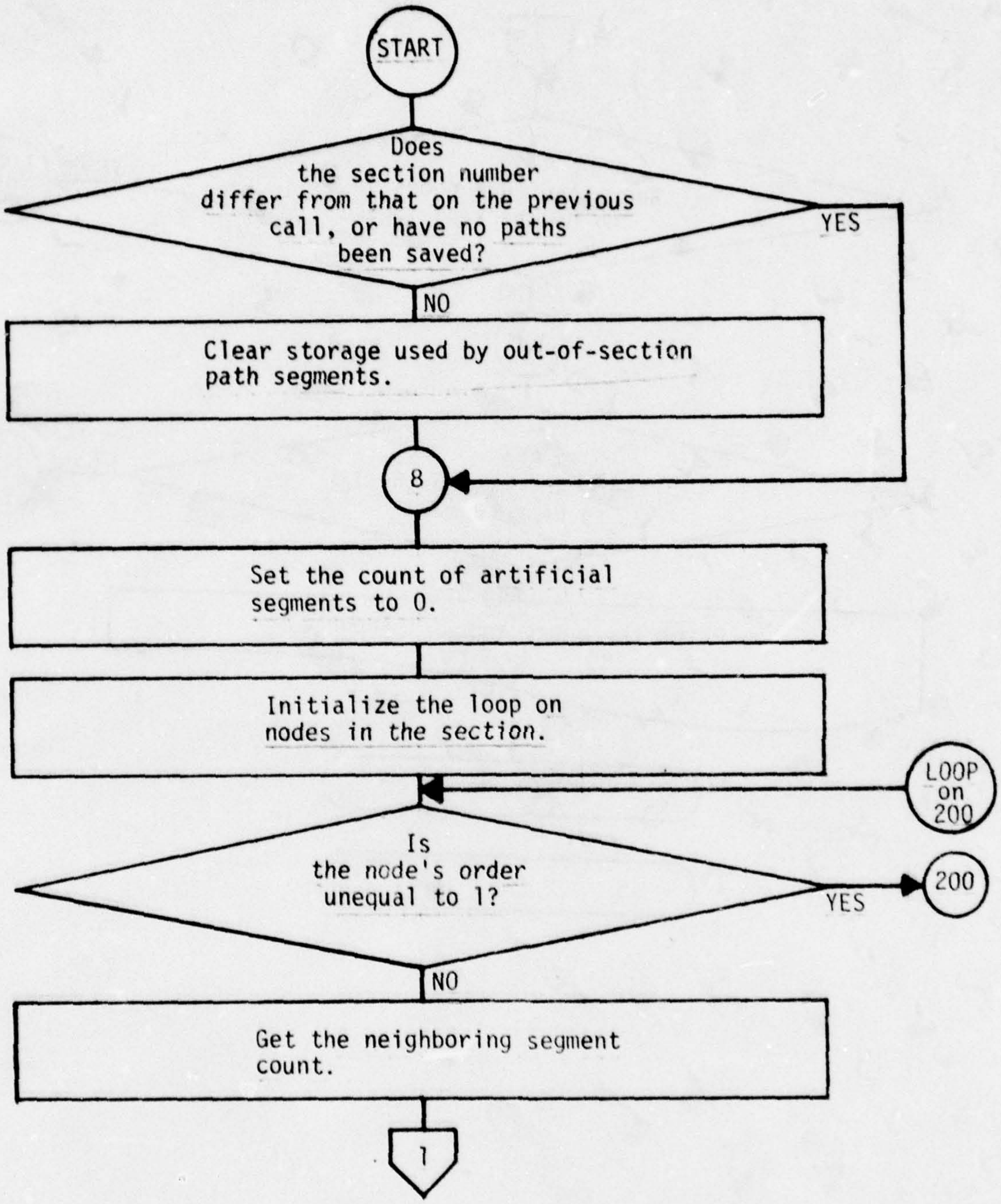
Subroutine EPXP



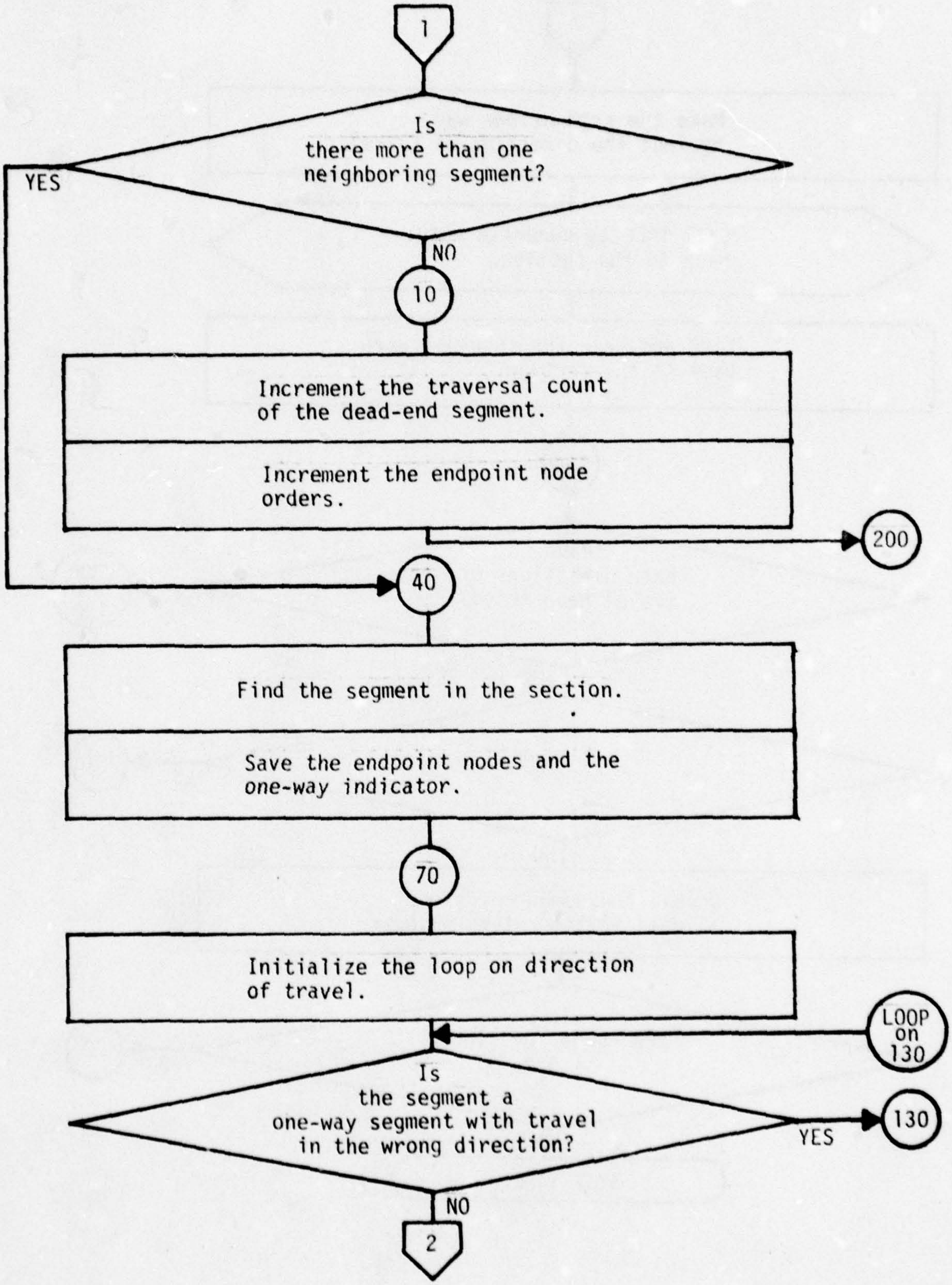
Subroutine EPXP



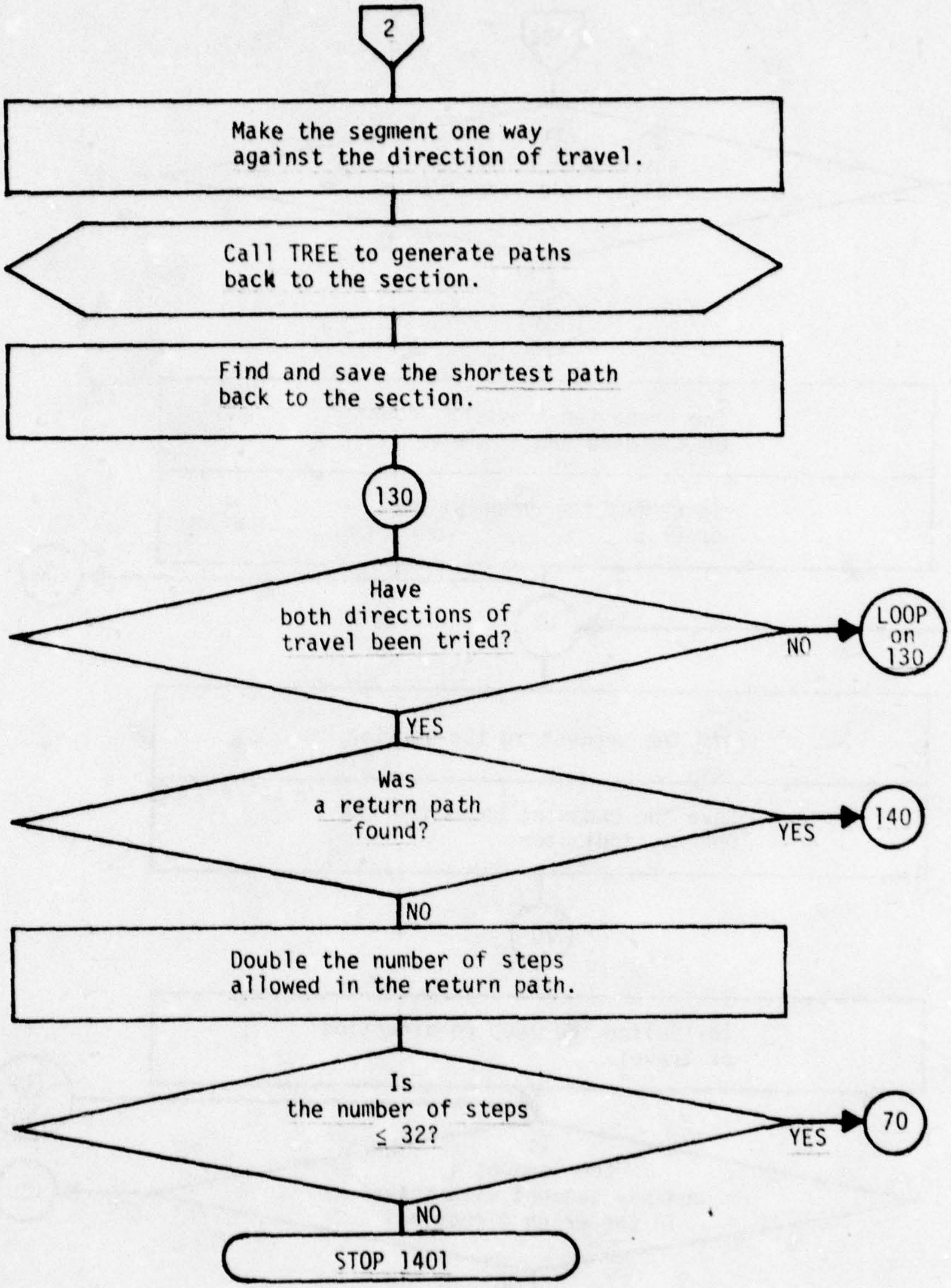
Subroutine EPXP



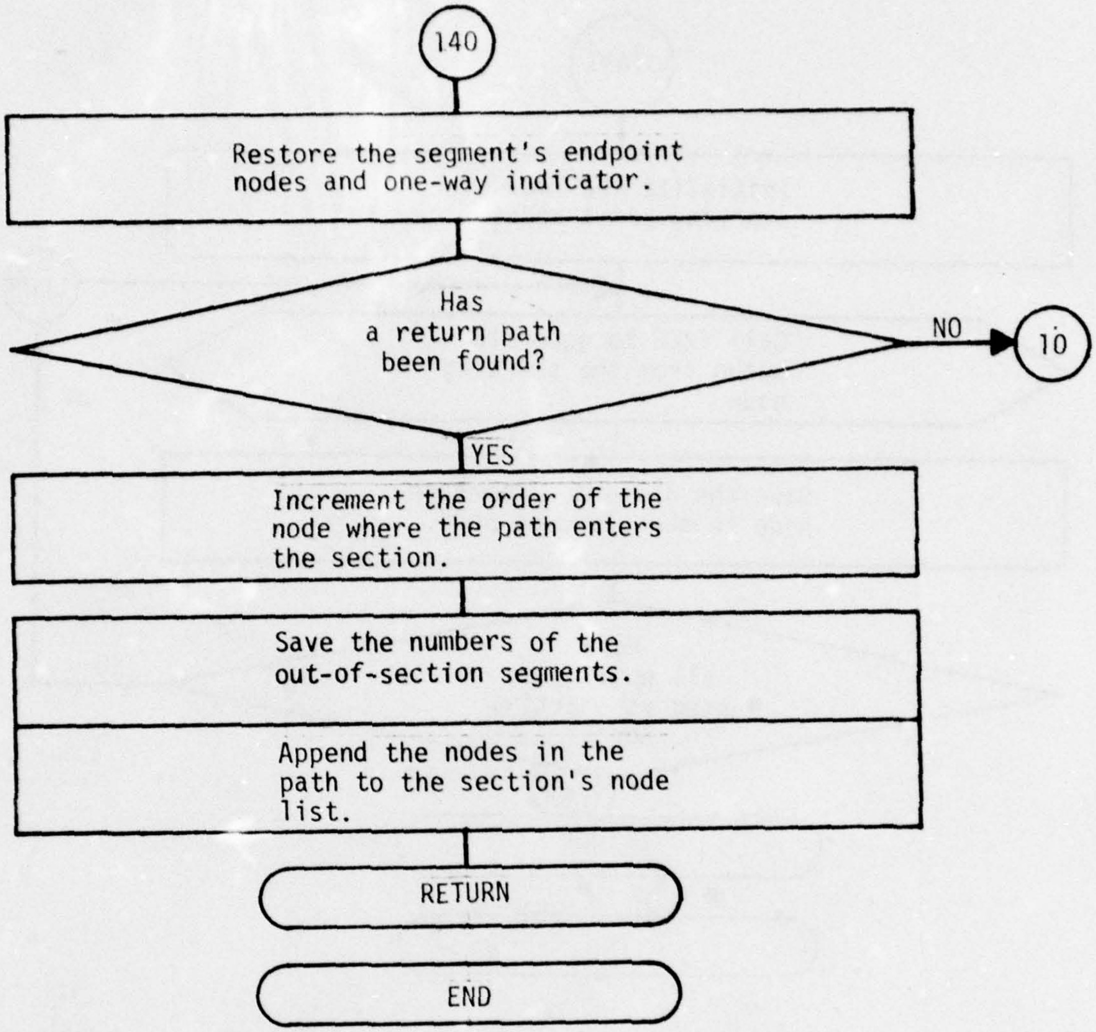
Subroutine CLOSE1



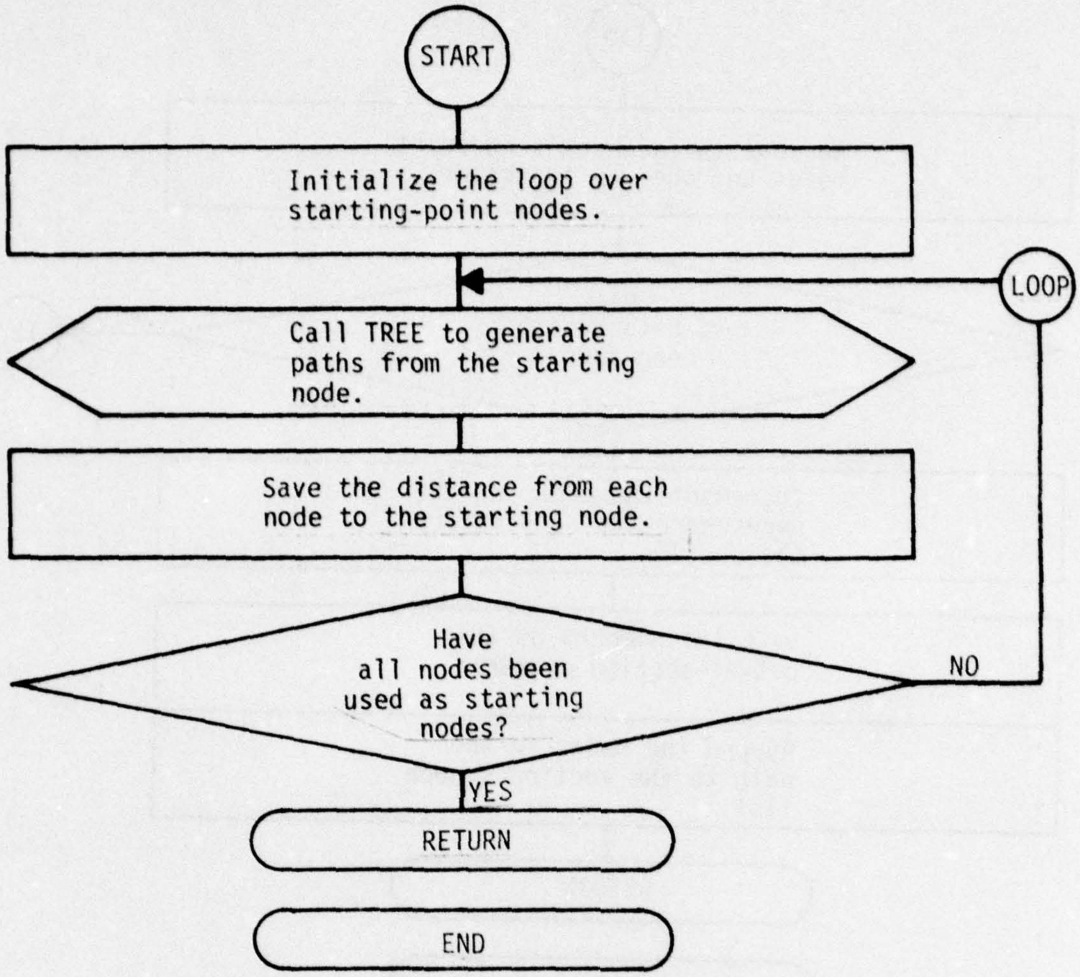
Subroutine CLOSE1



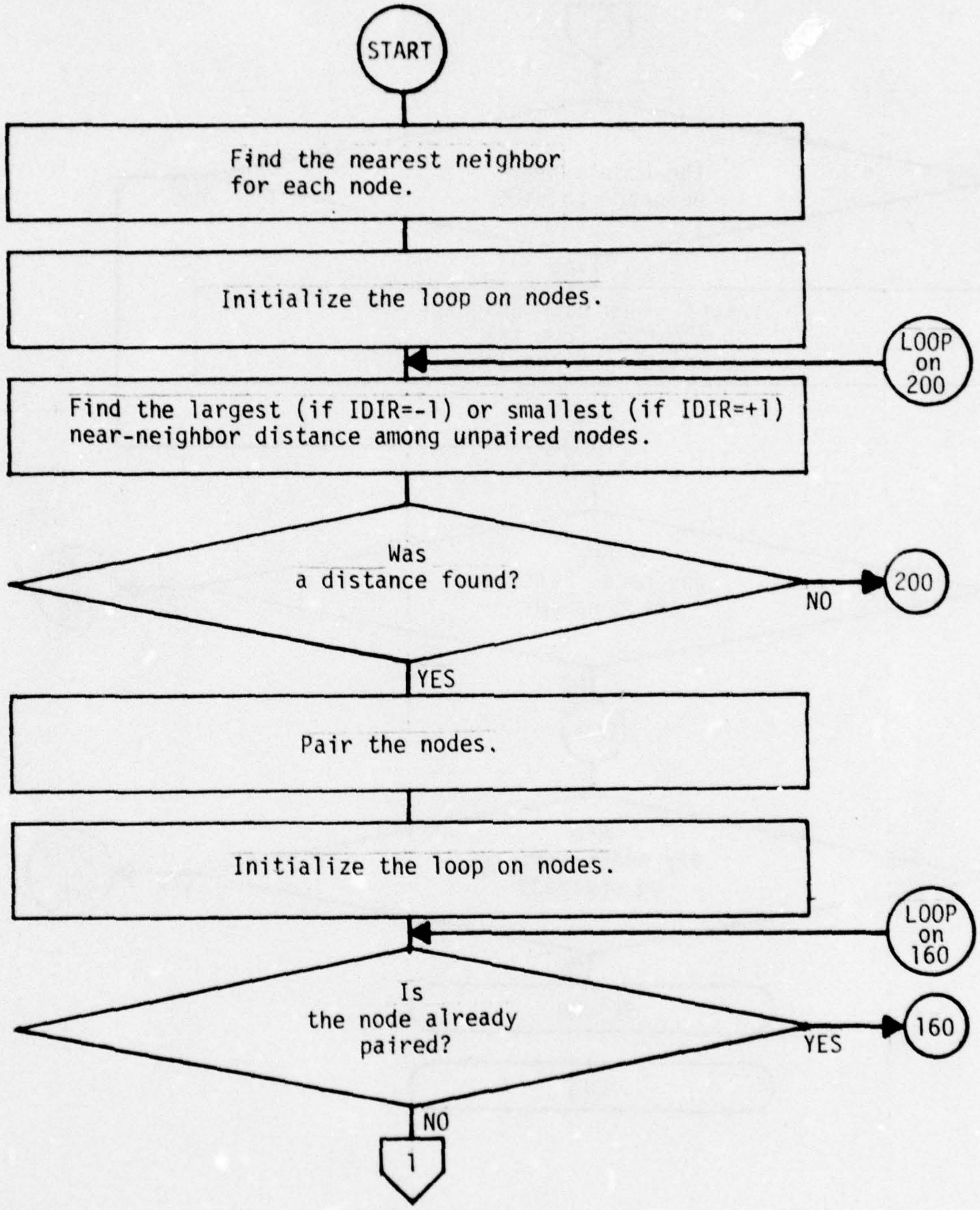
Subroutine CLOSE1



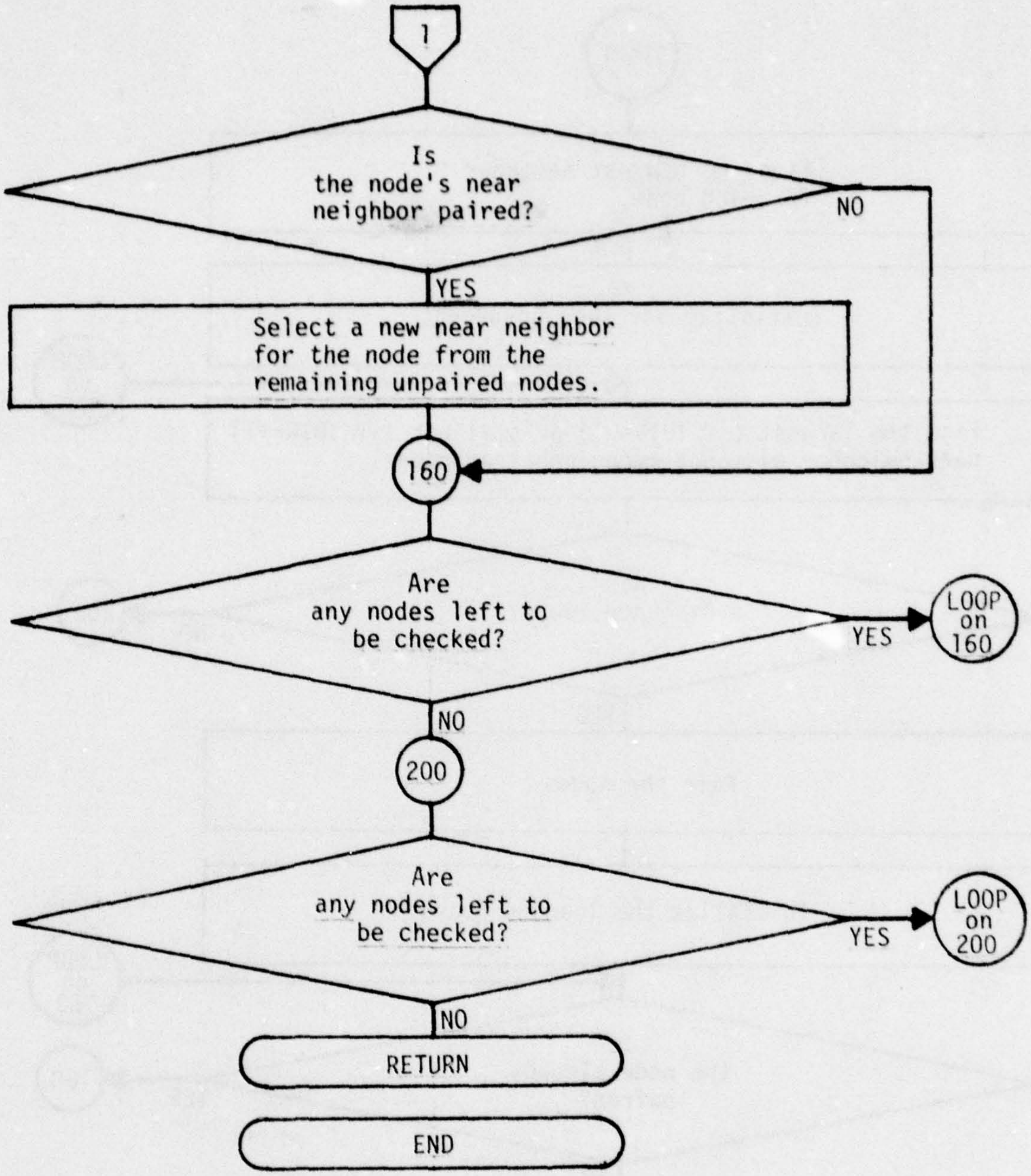
Subroutine CLOSE1



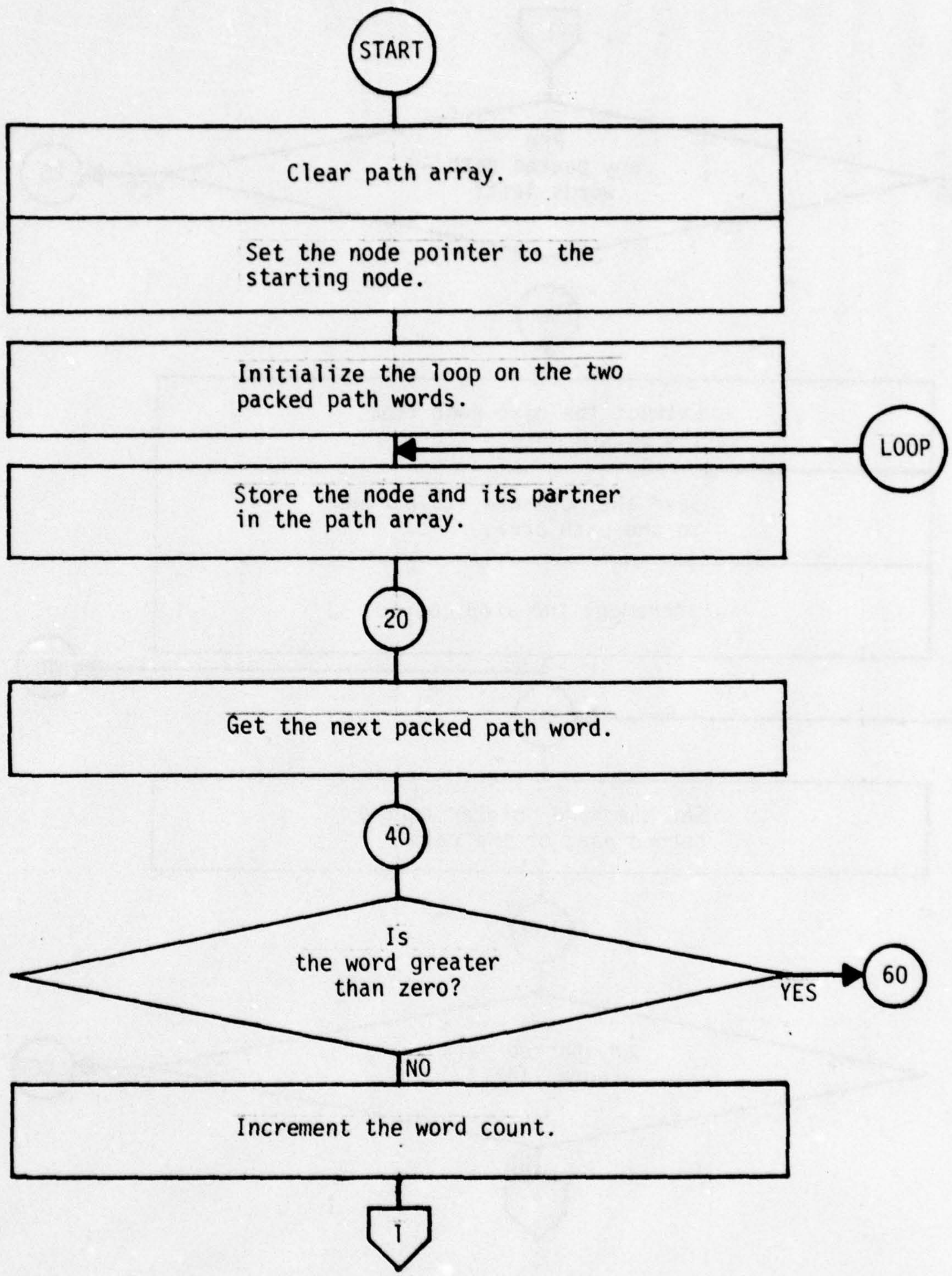
Subroutine GENDM



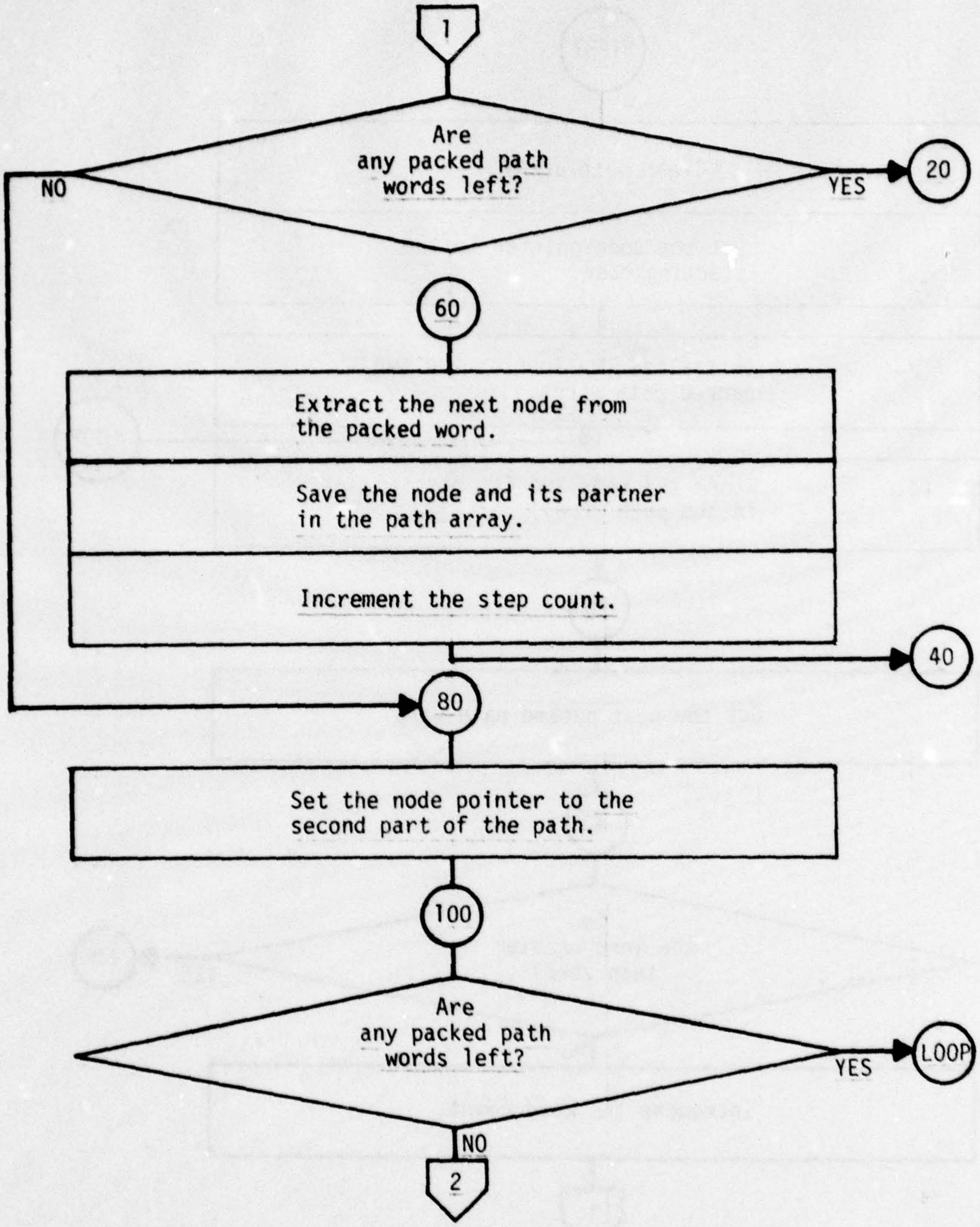
Subroutine SELORD



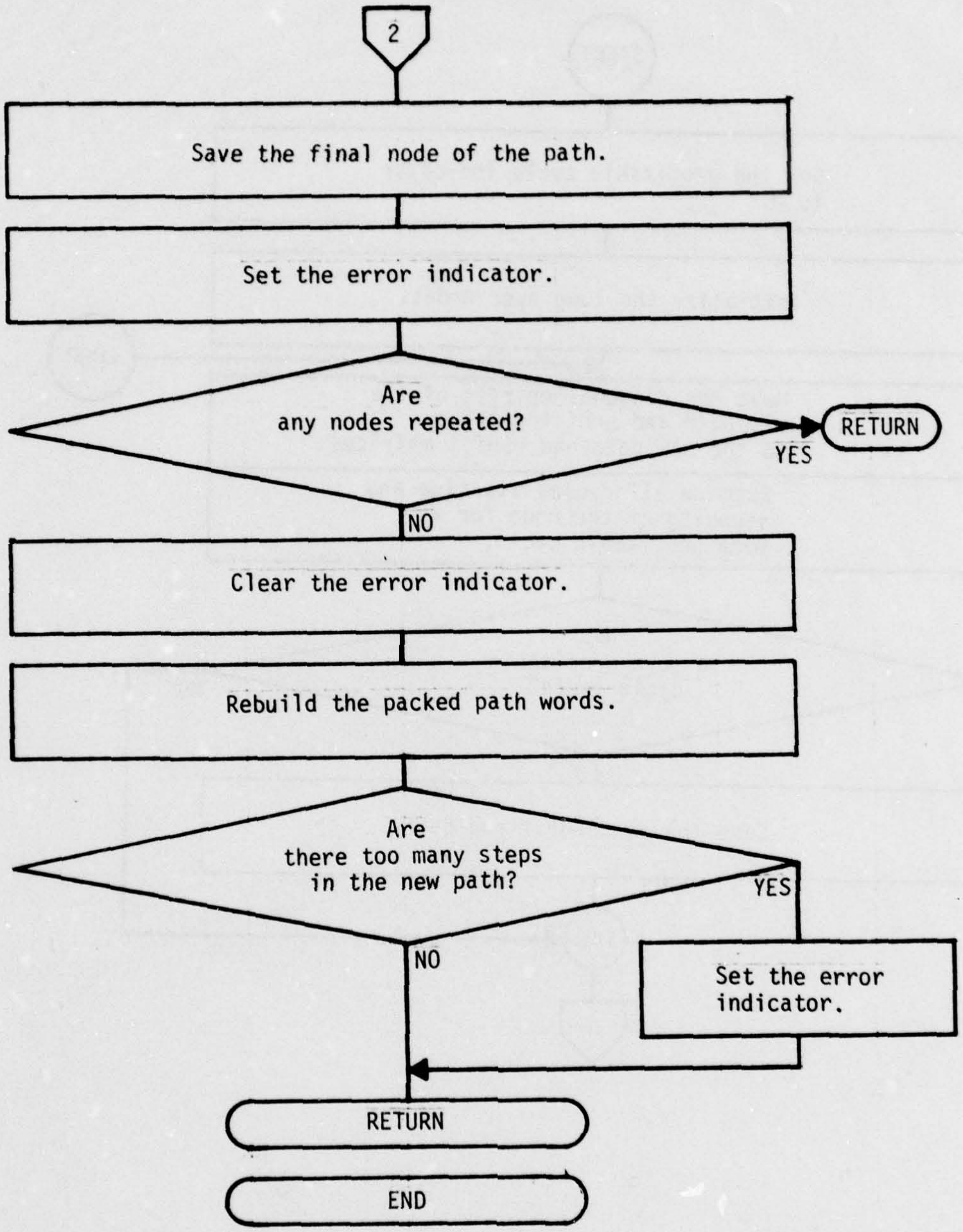
Subroutine SELORD



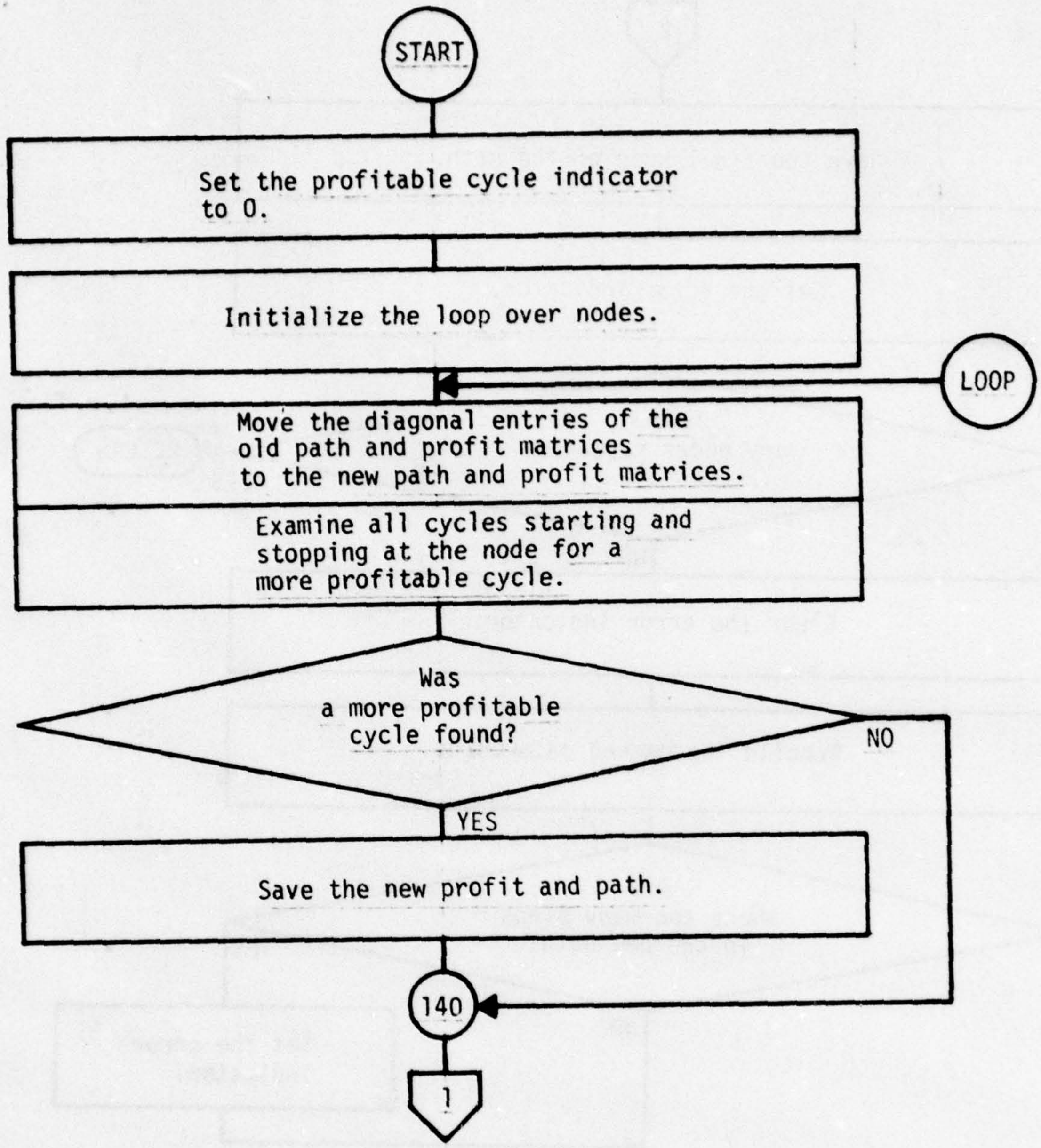
Subroutine PATH



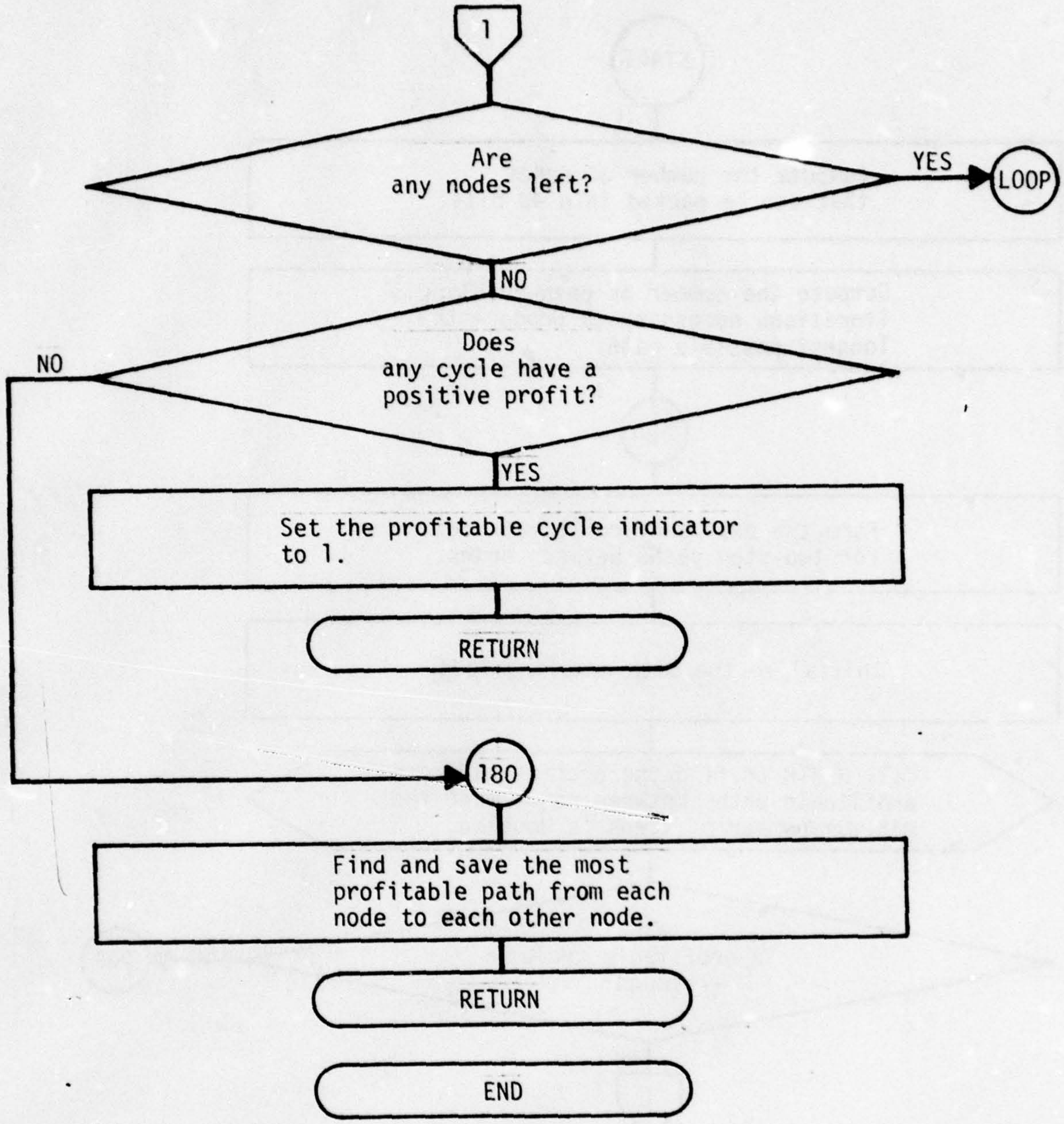
Subroutine PATH



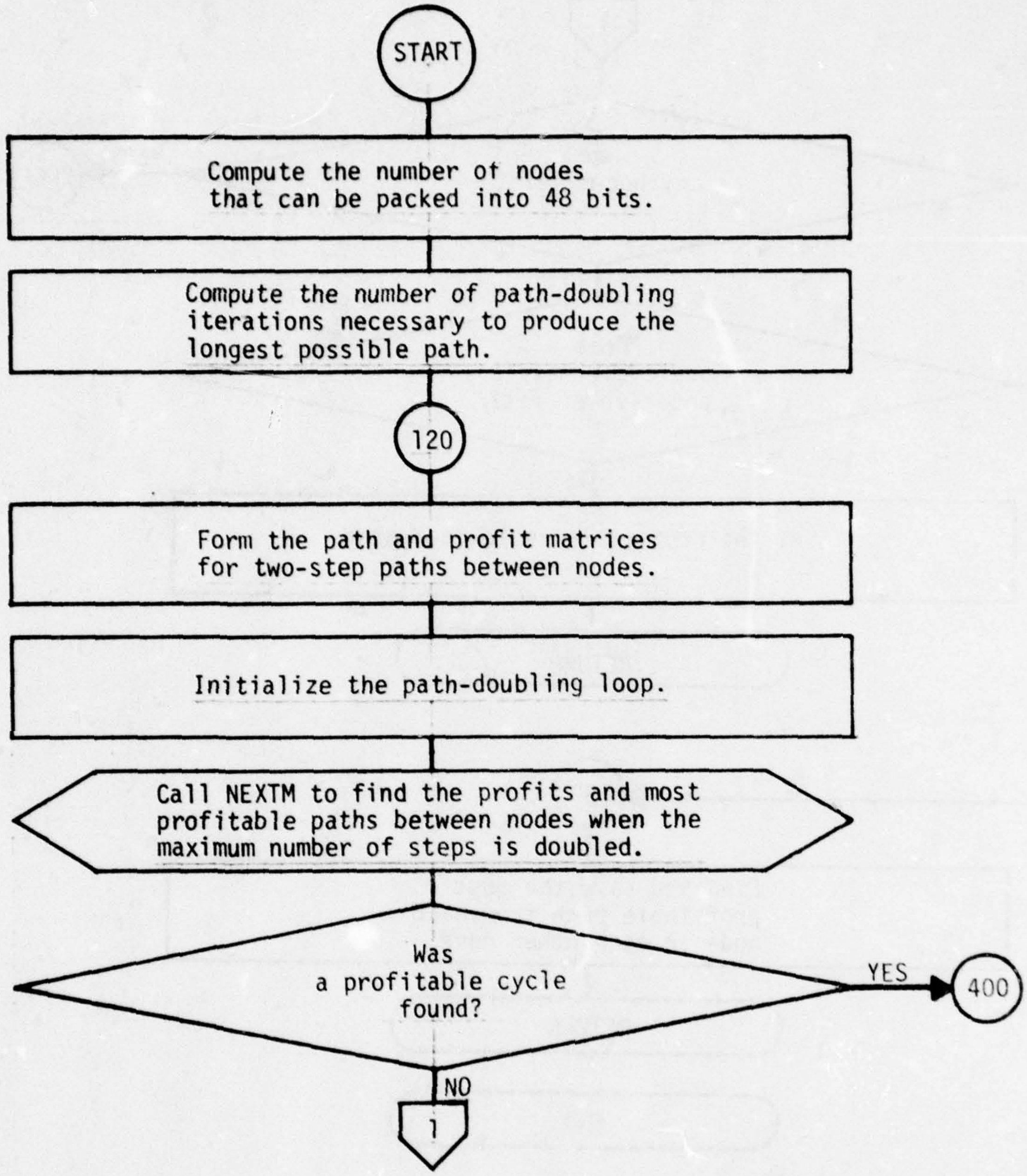
Subroutine PATH



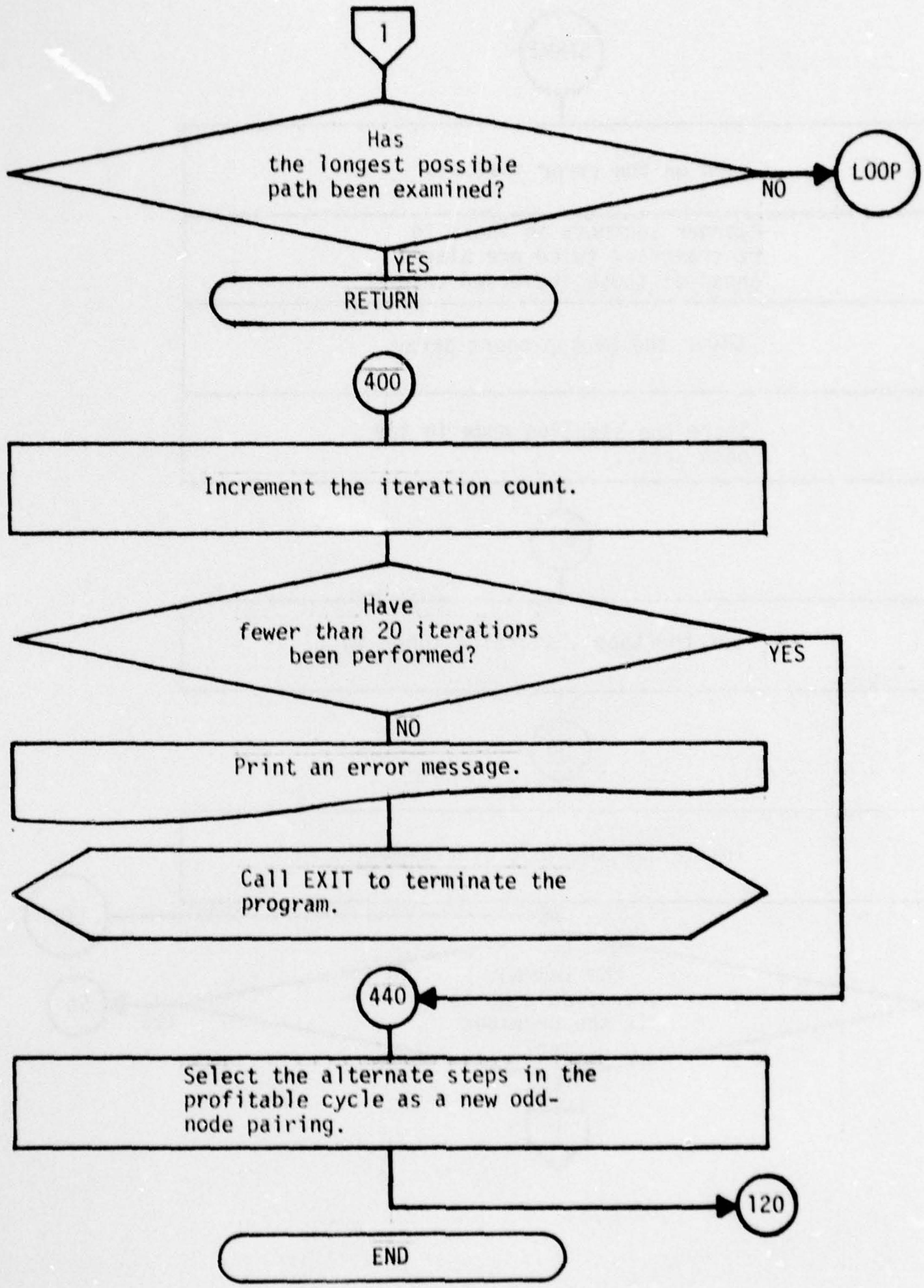
Subroutine NEXTM



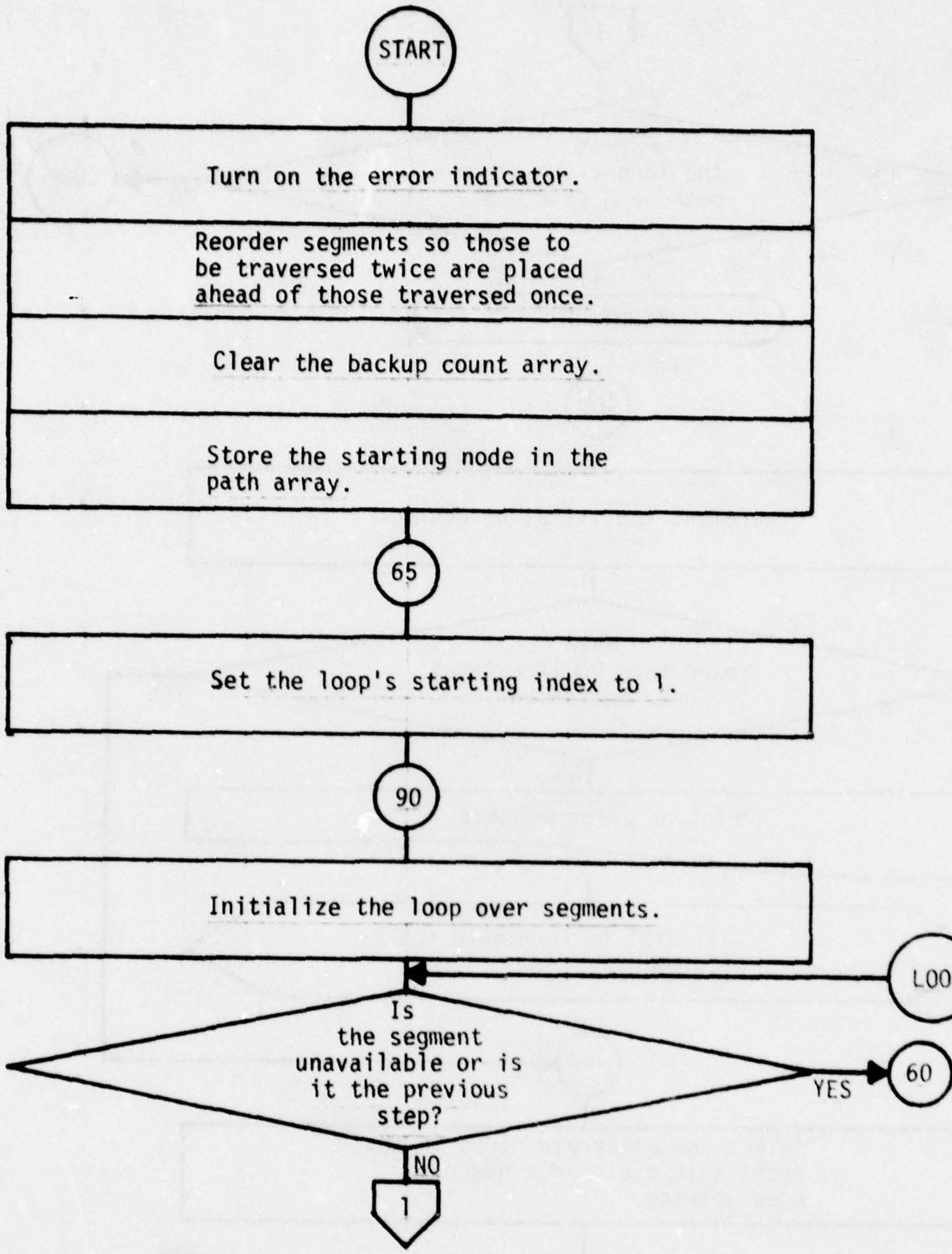
Subroutine NEXTM



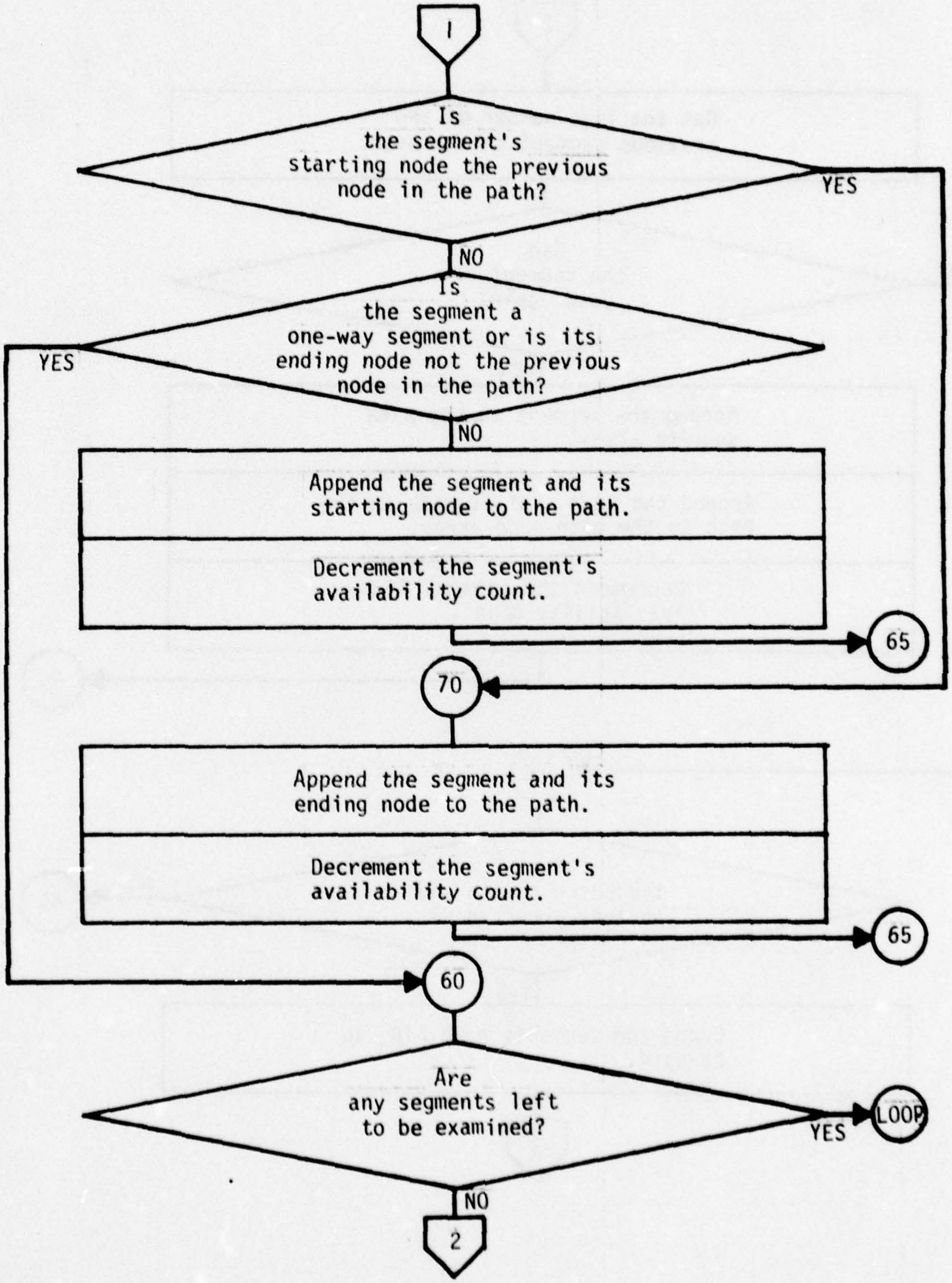
Subroutine SOLV



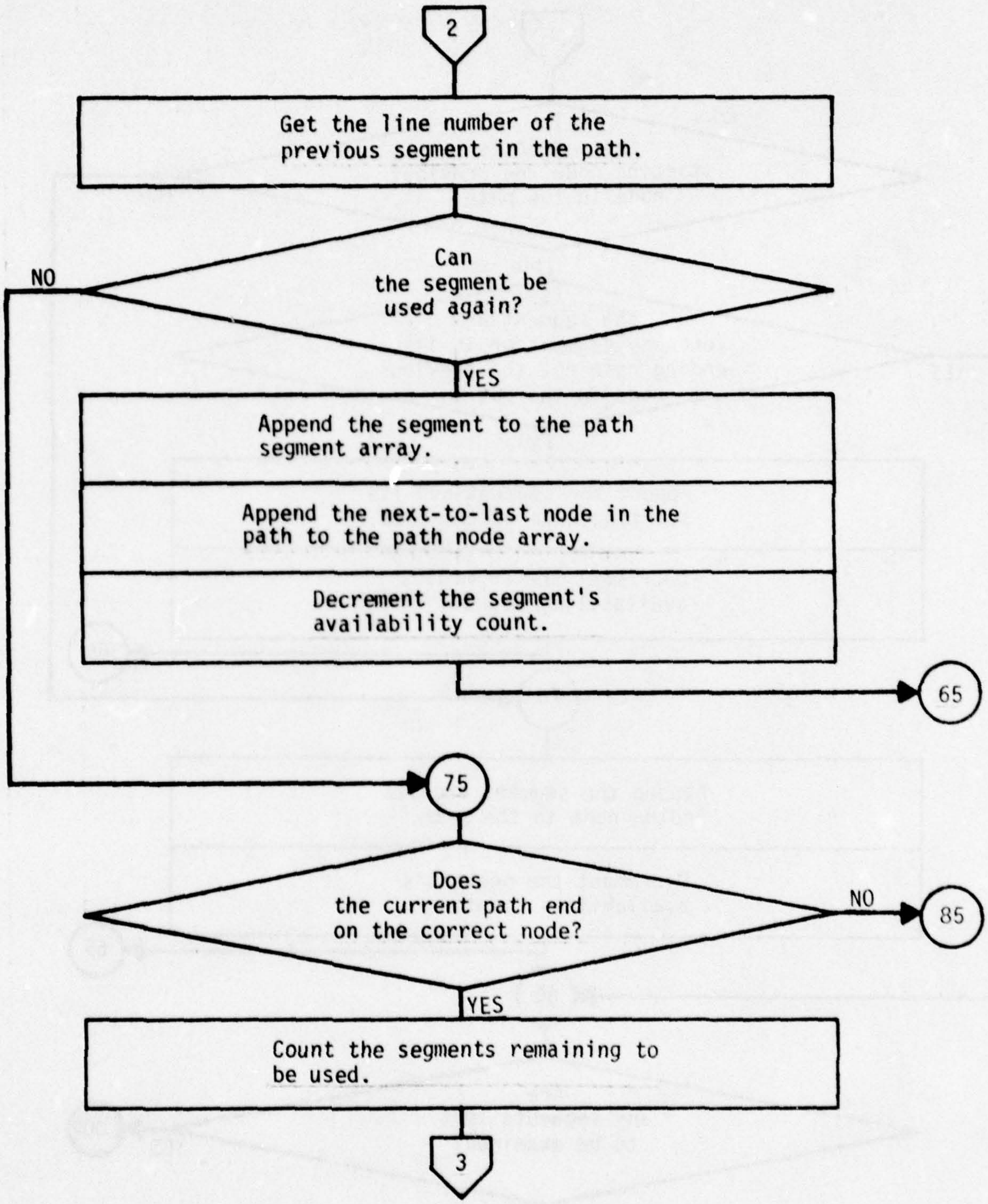
Subroutine SOLV



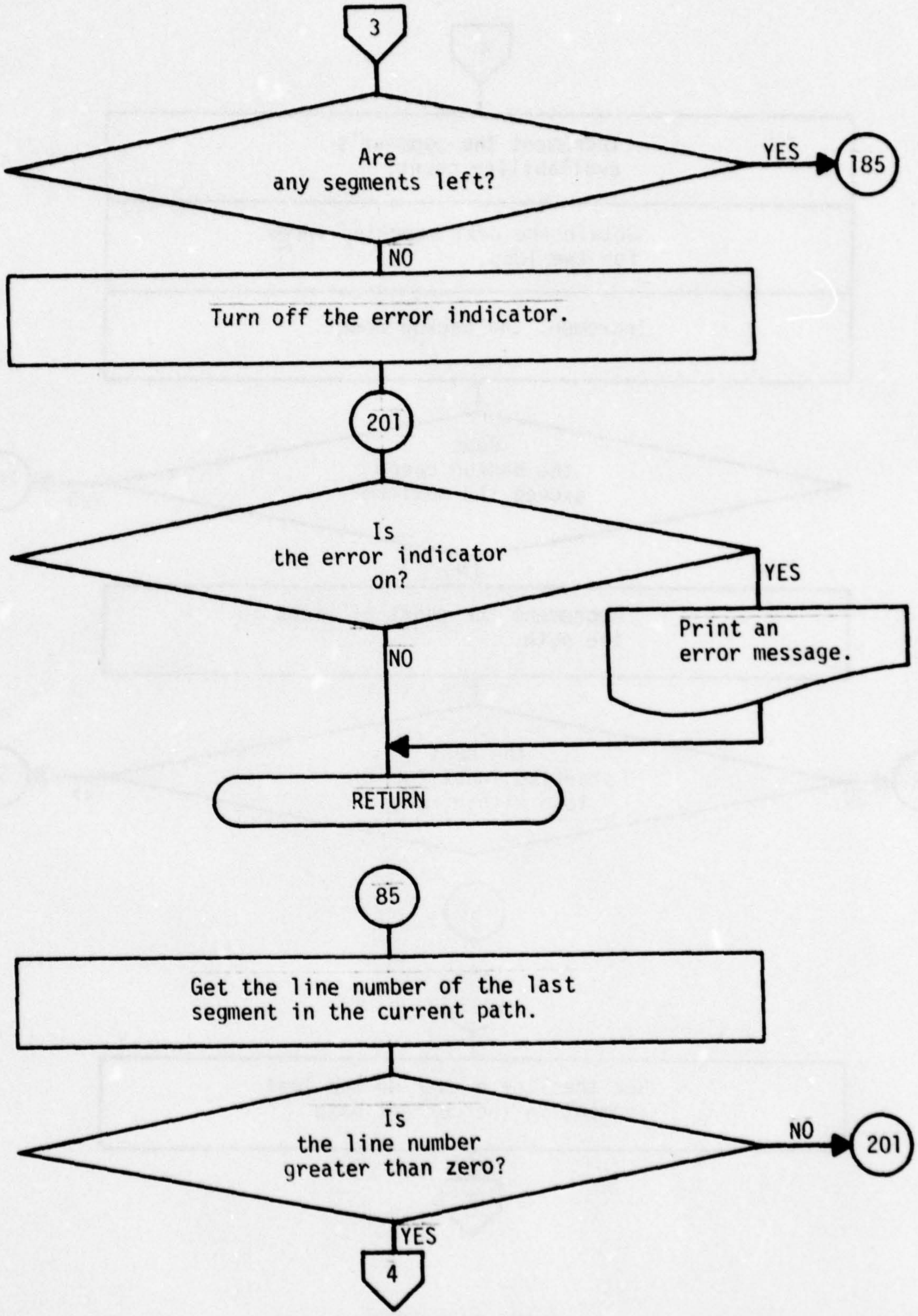
Subroutine TRAVEL



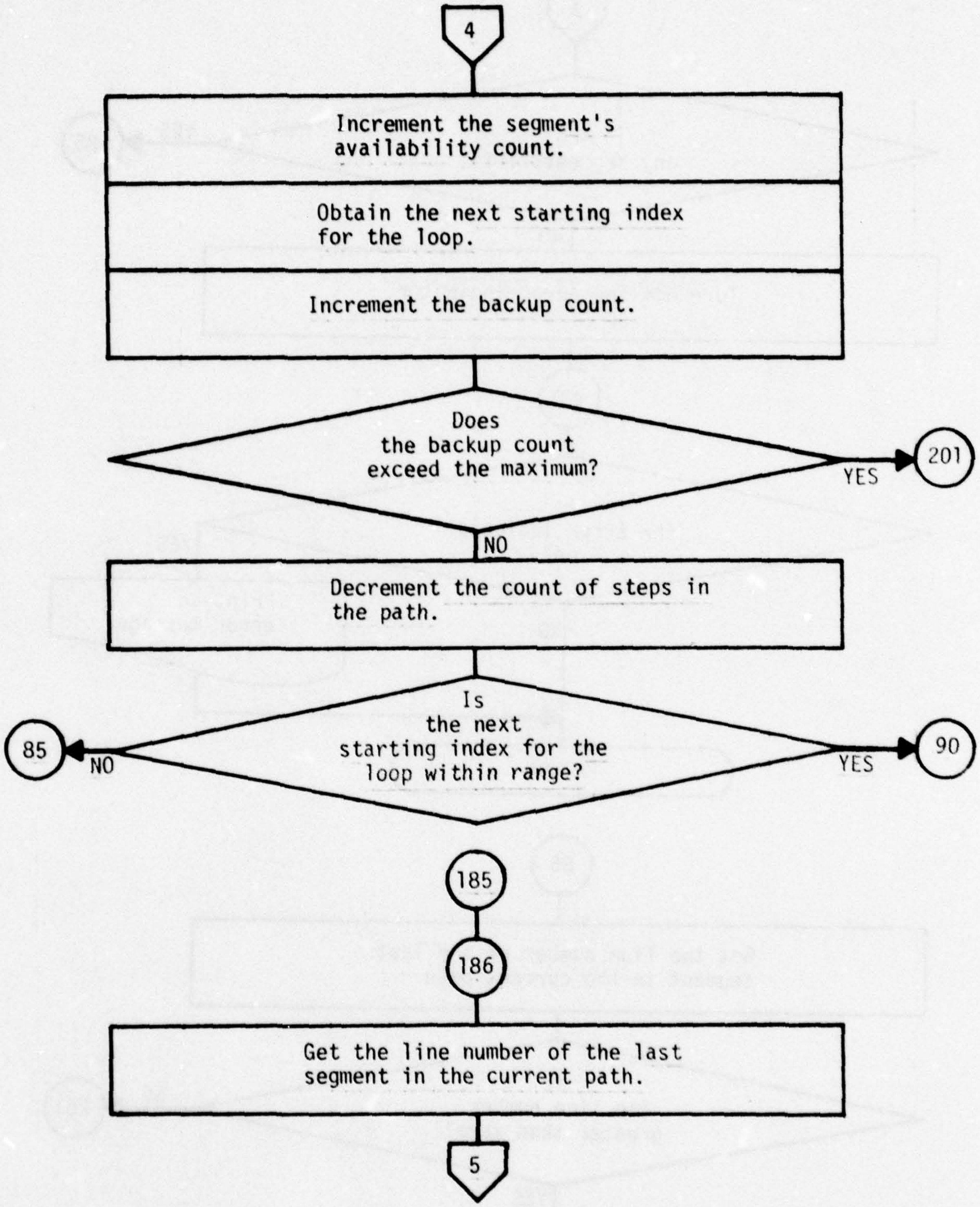
Subroutine TRAVEL



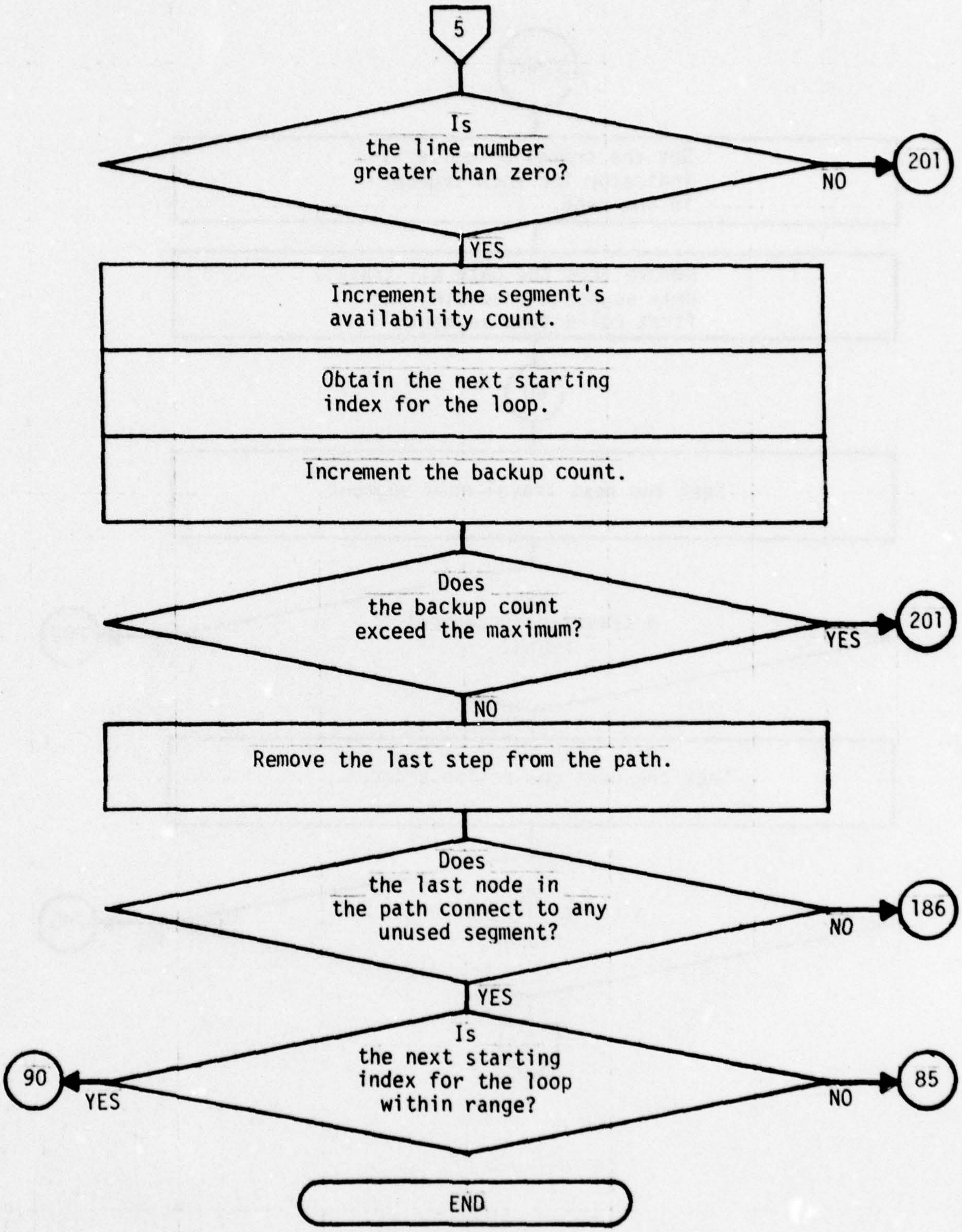
Subroutine TRAVEL



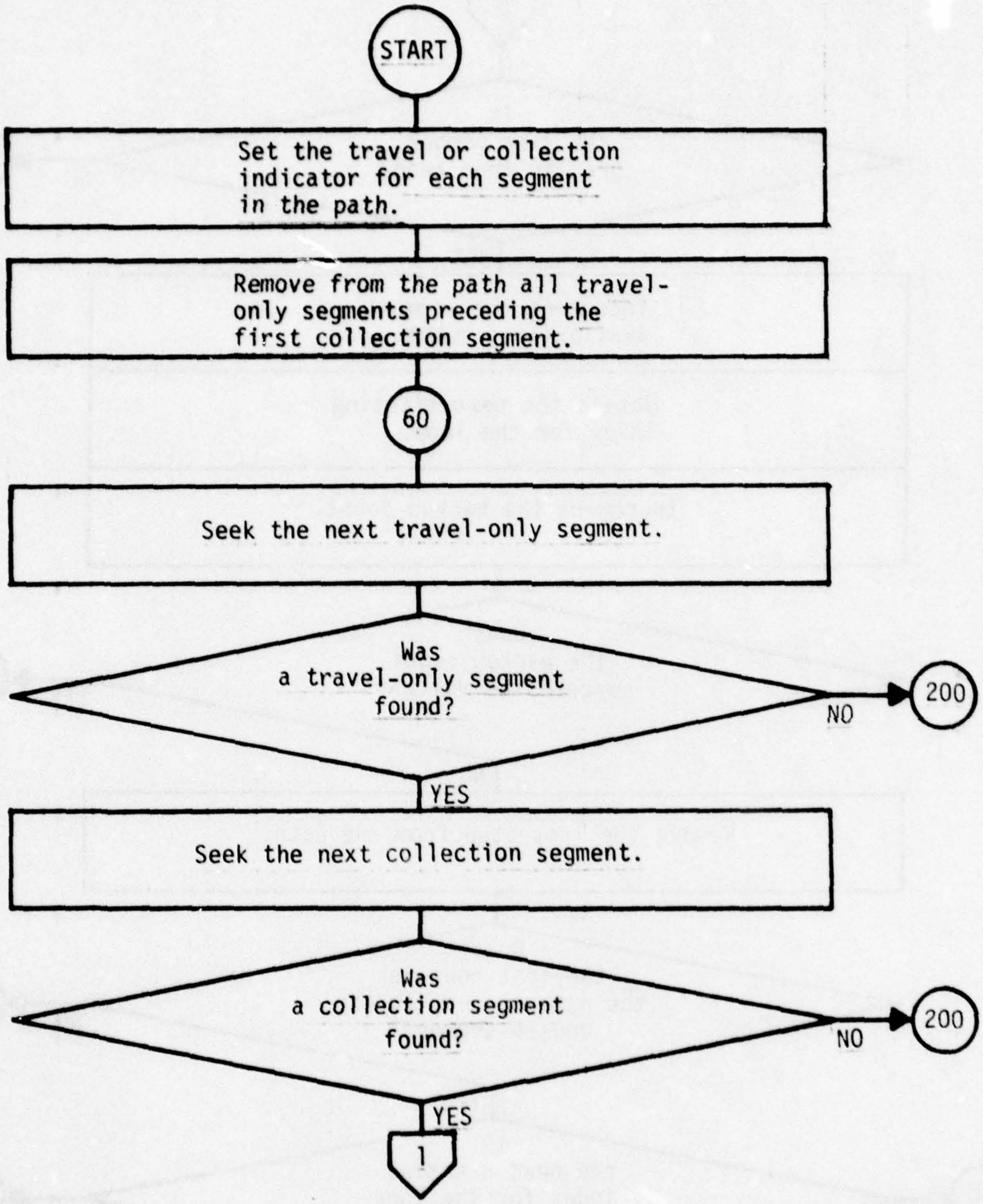
Subroutine TRAVEL



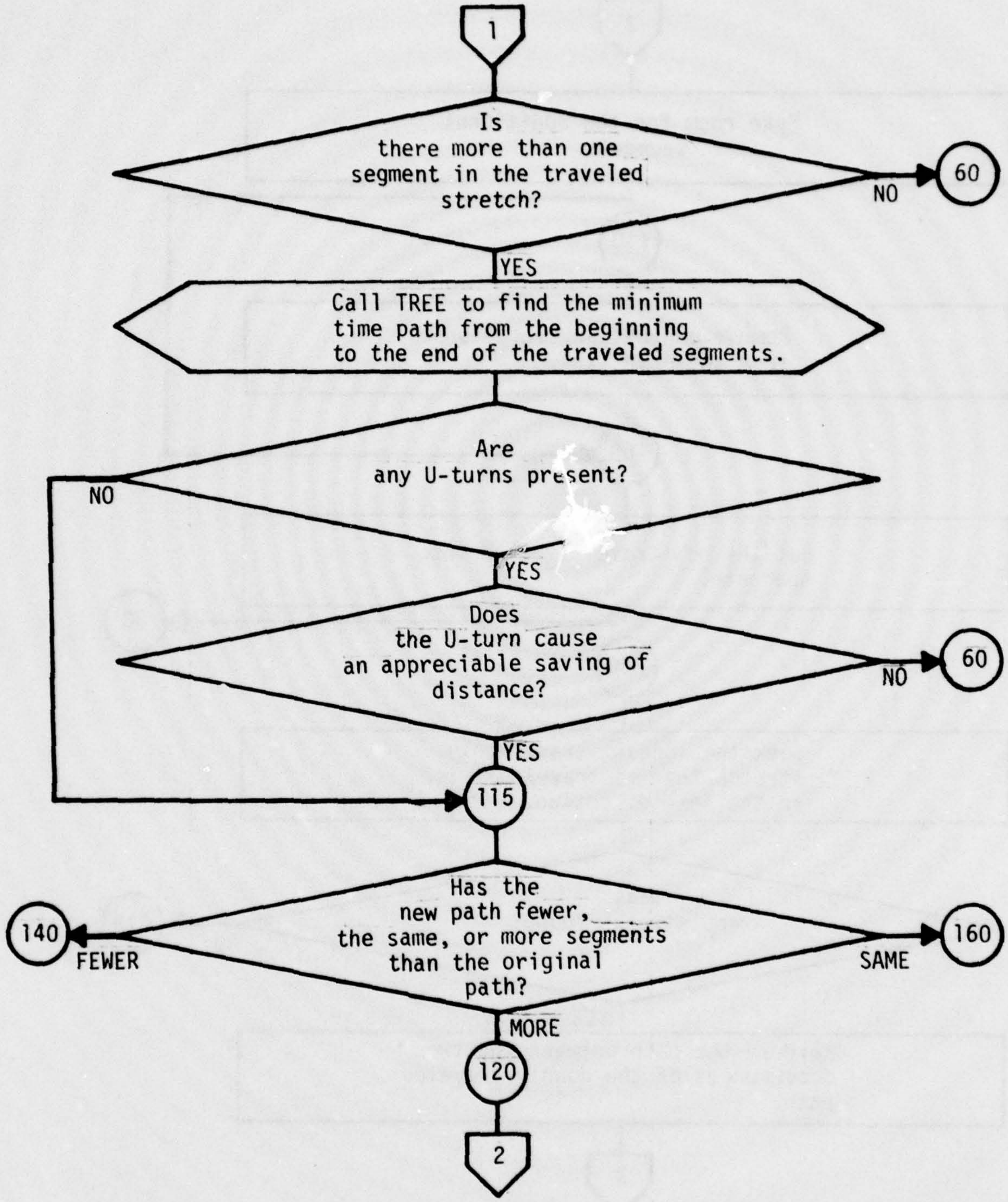
Subroutine TRAVEL



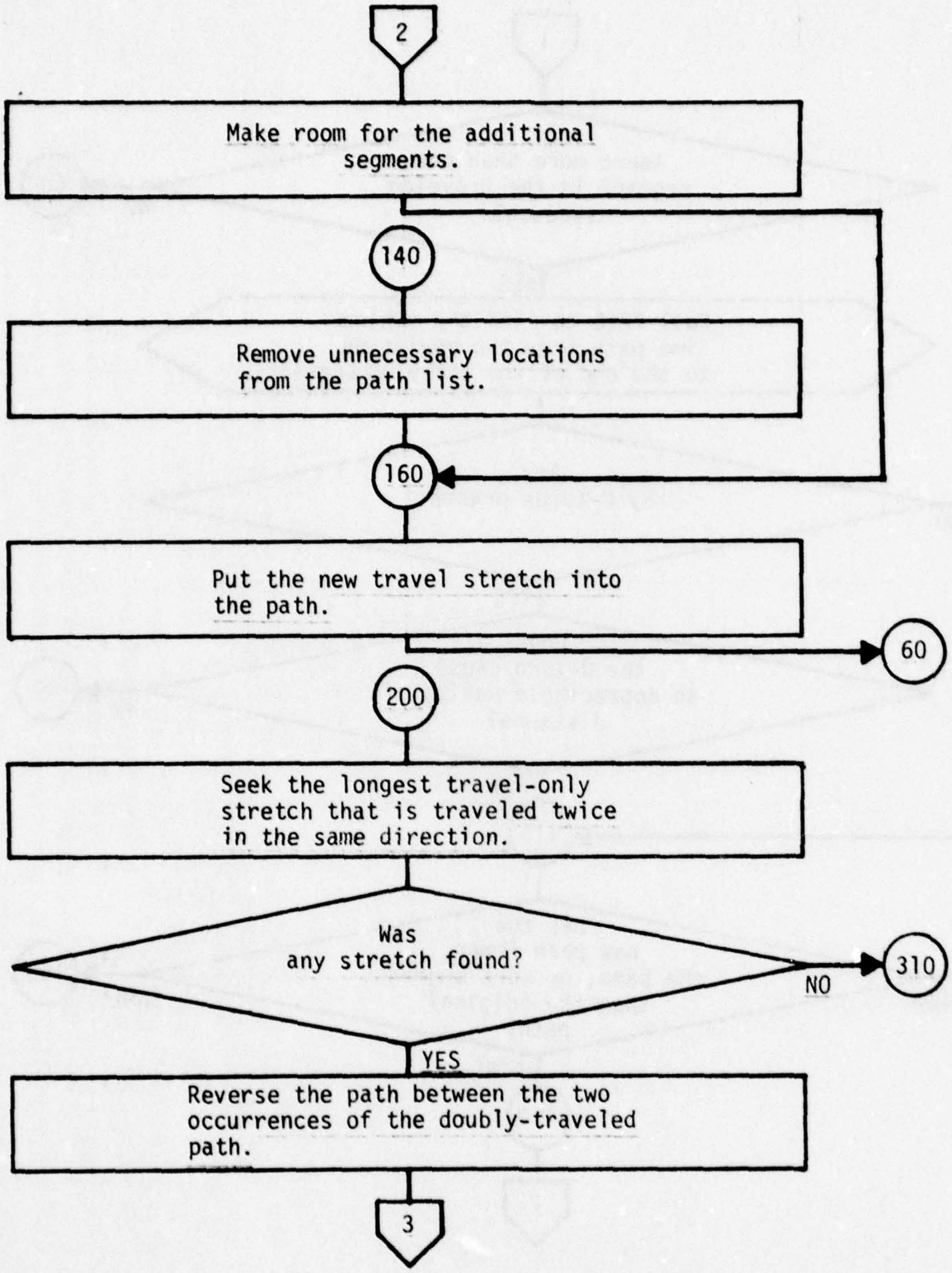
Subroutine TRAVEL



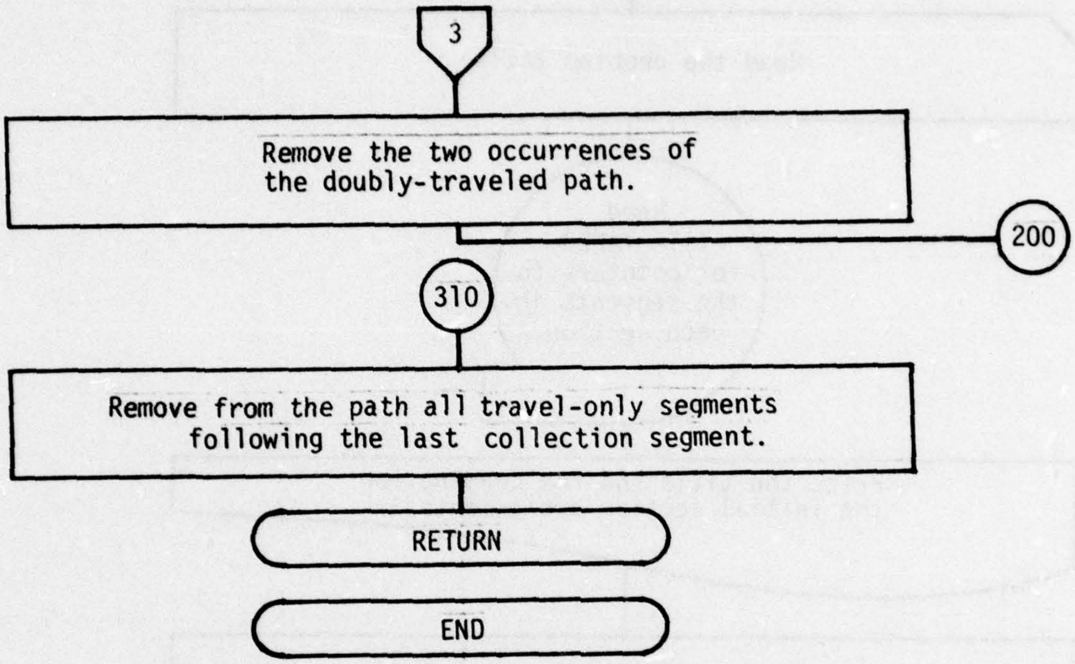
Subroutine OPTPATH



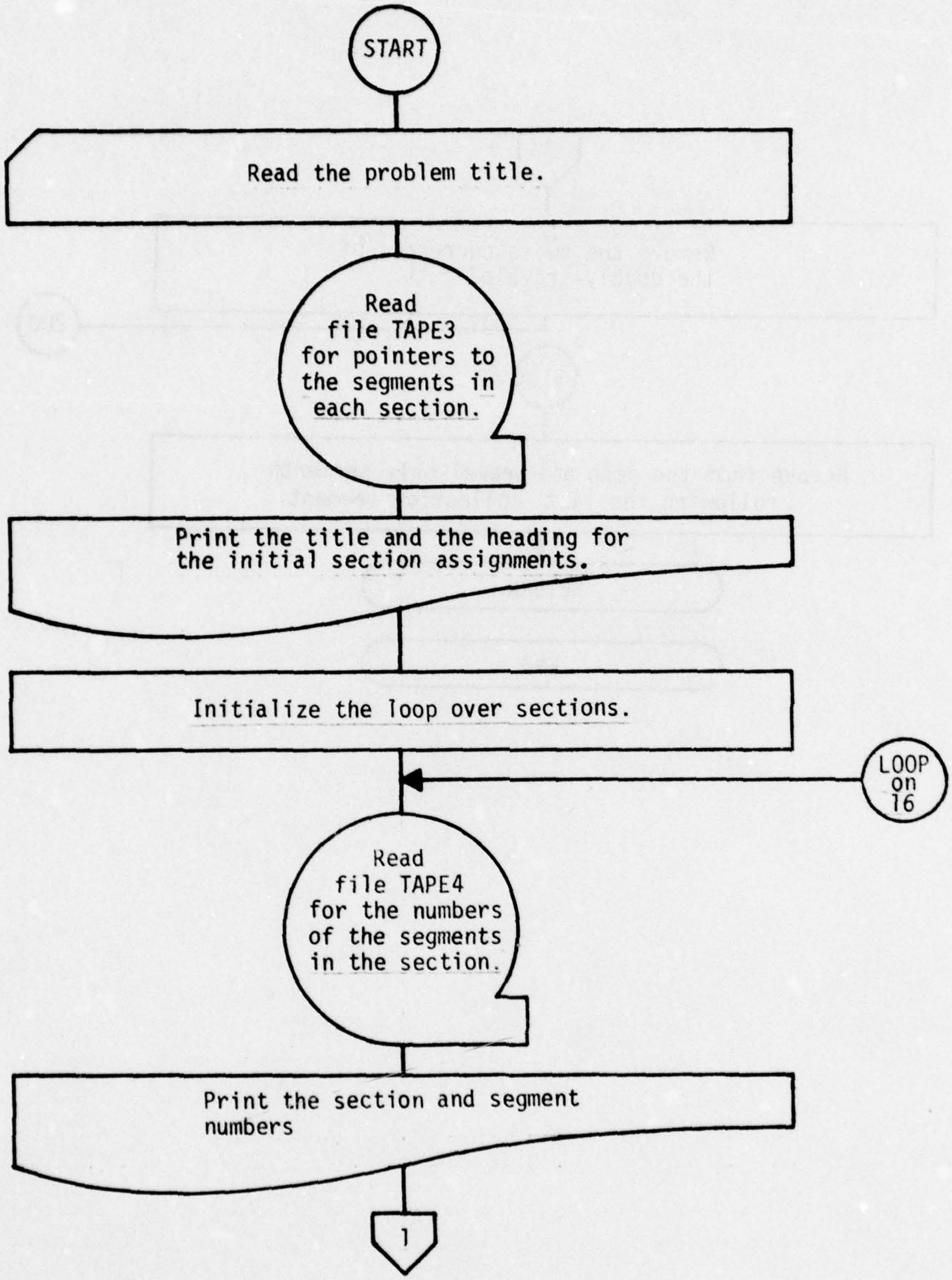
Subroutine OPTPATH



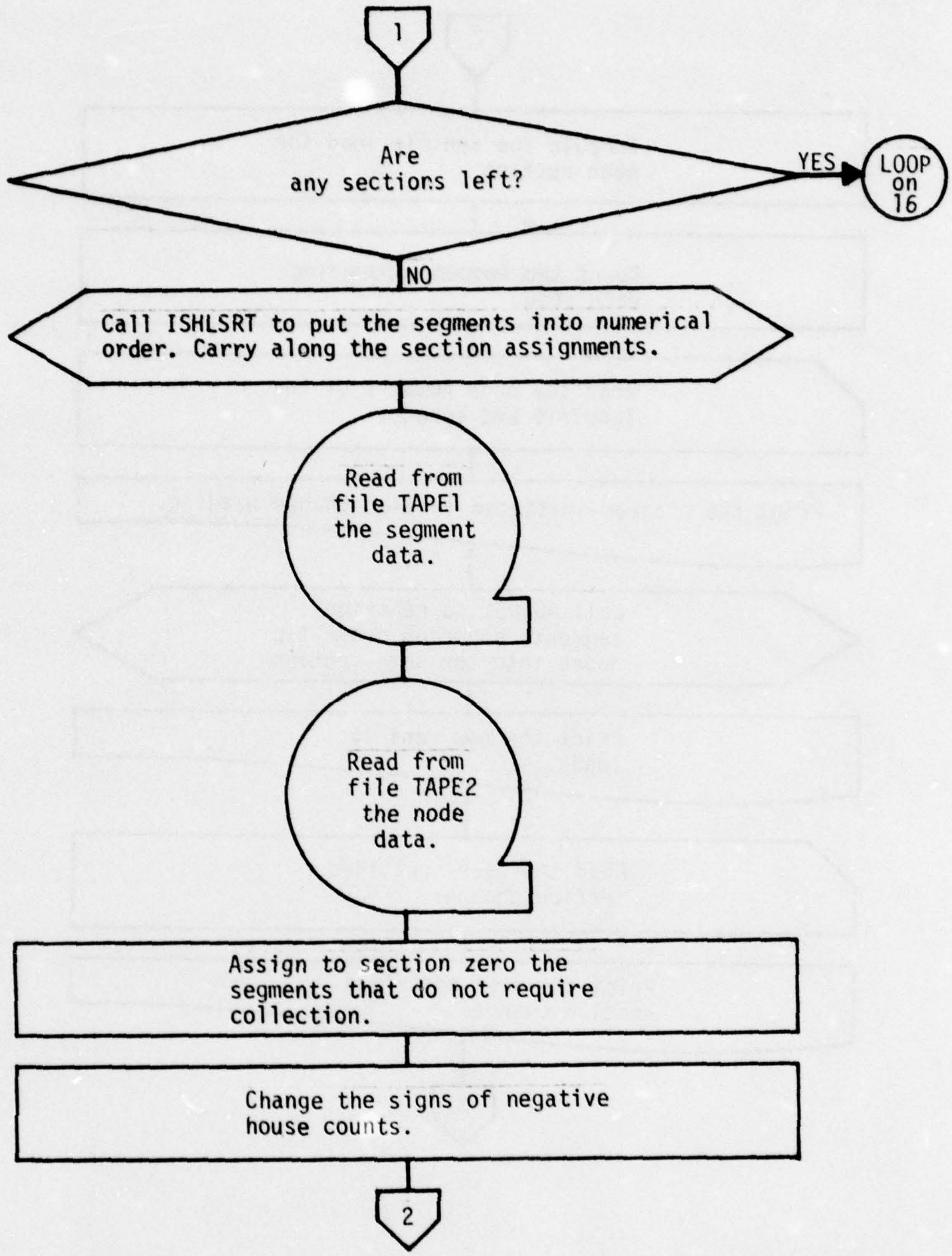
Subroutine OPTPATH



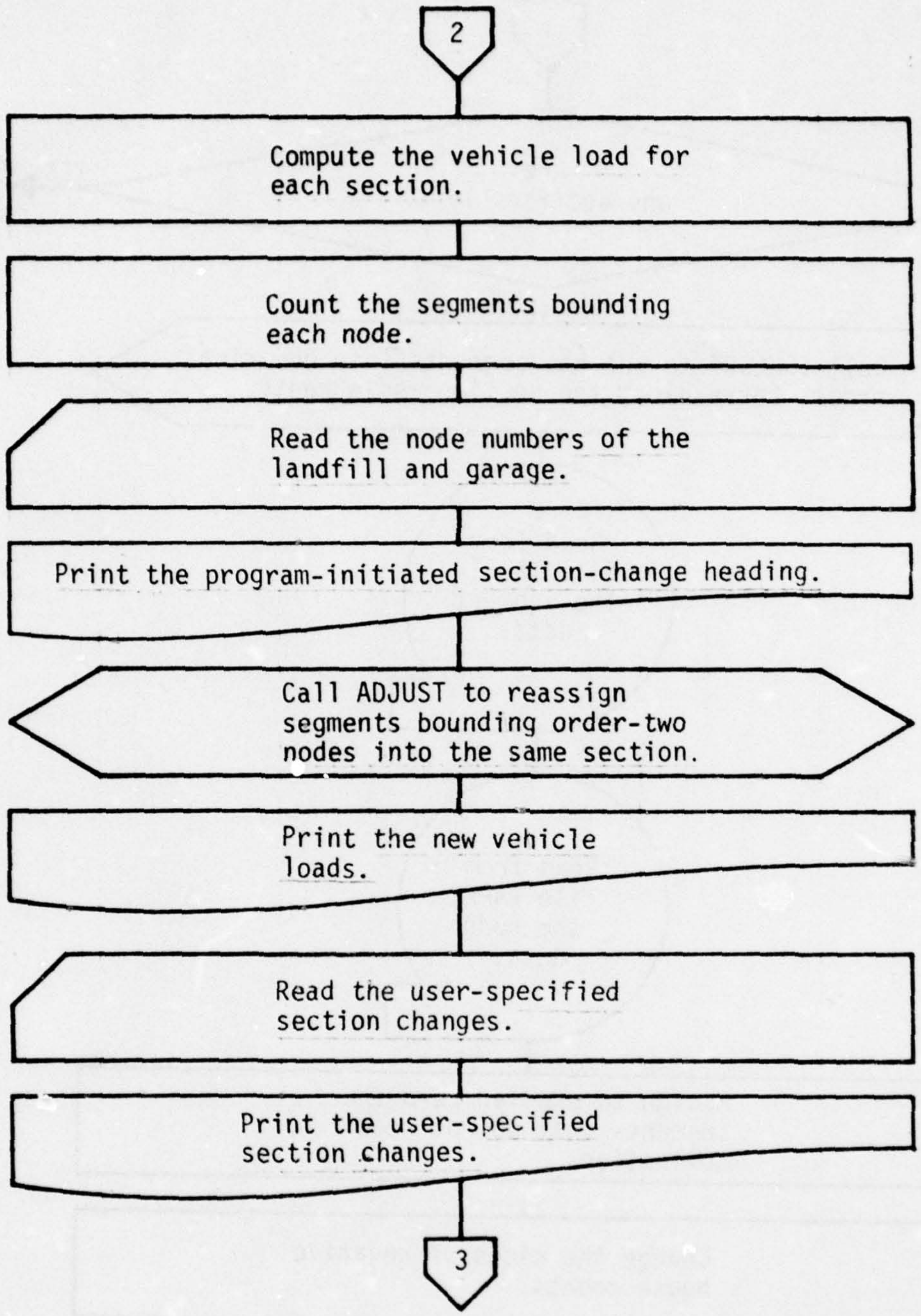
Subroutine OPTPATH



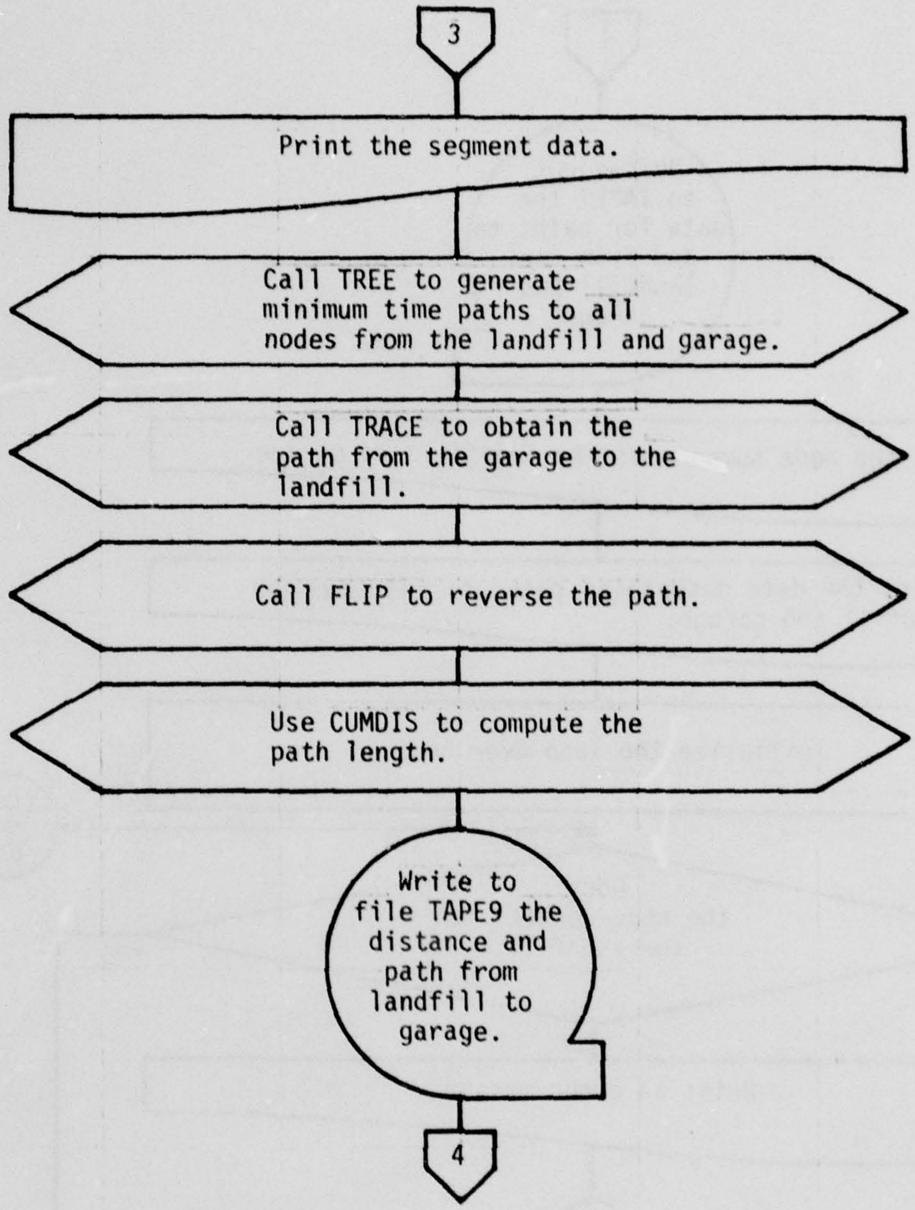
Program PHASE3



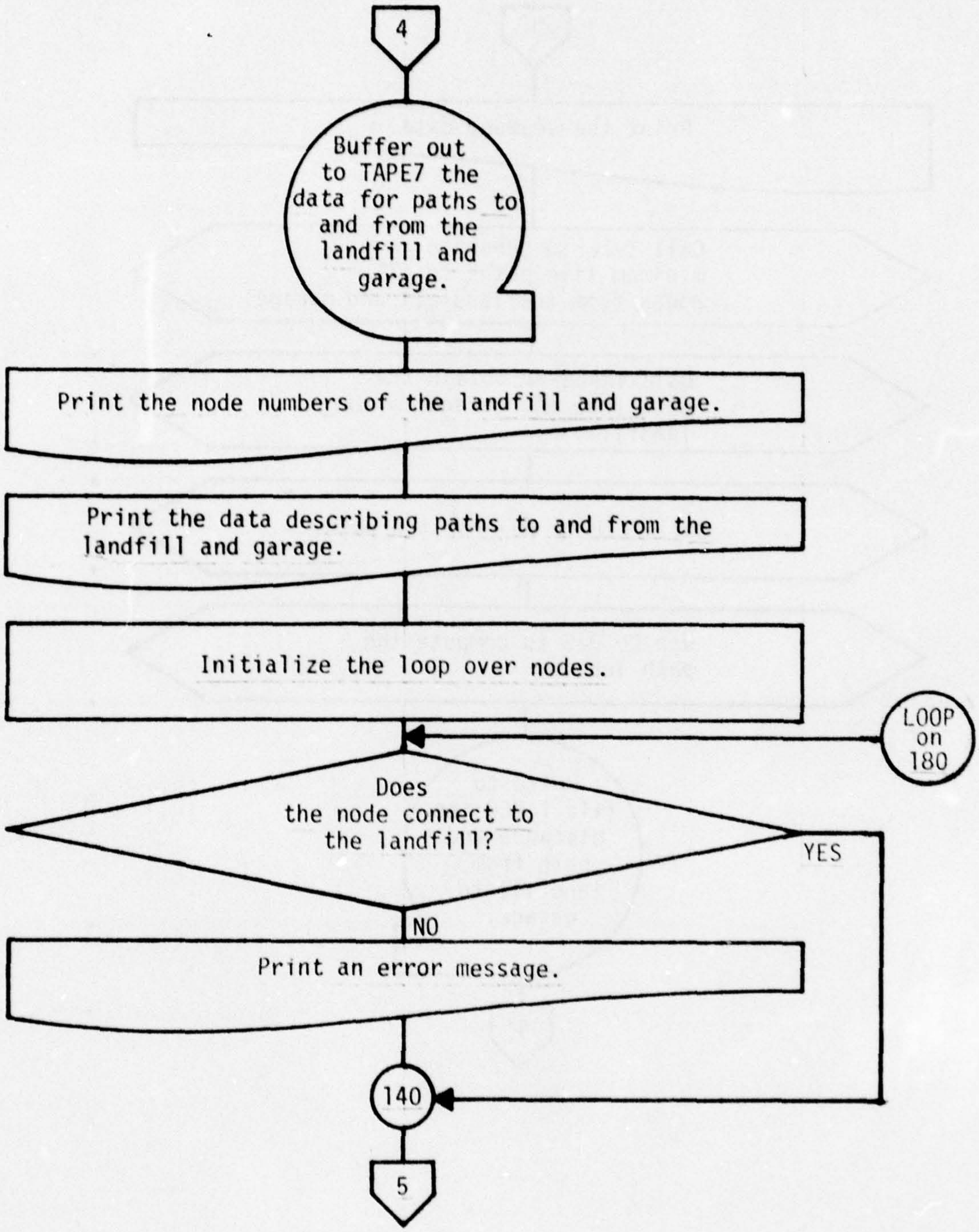
Program PHASE3



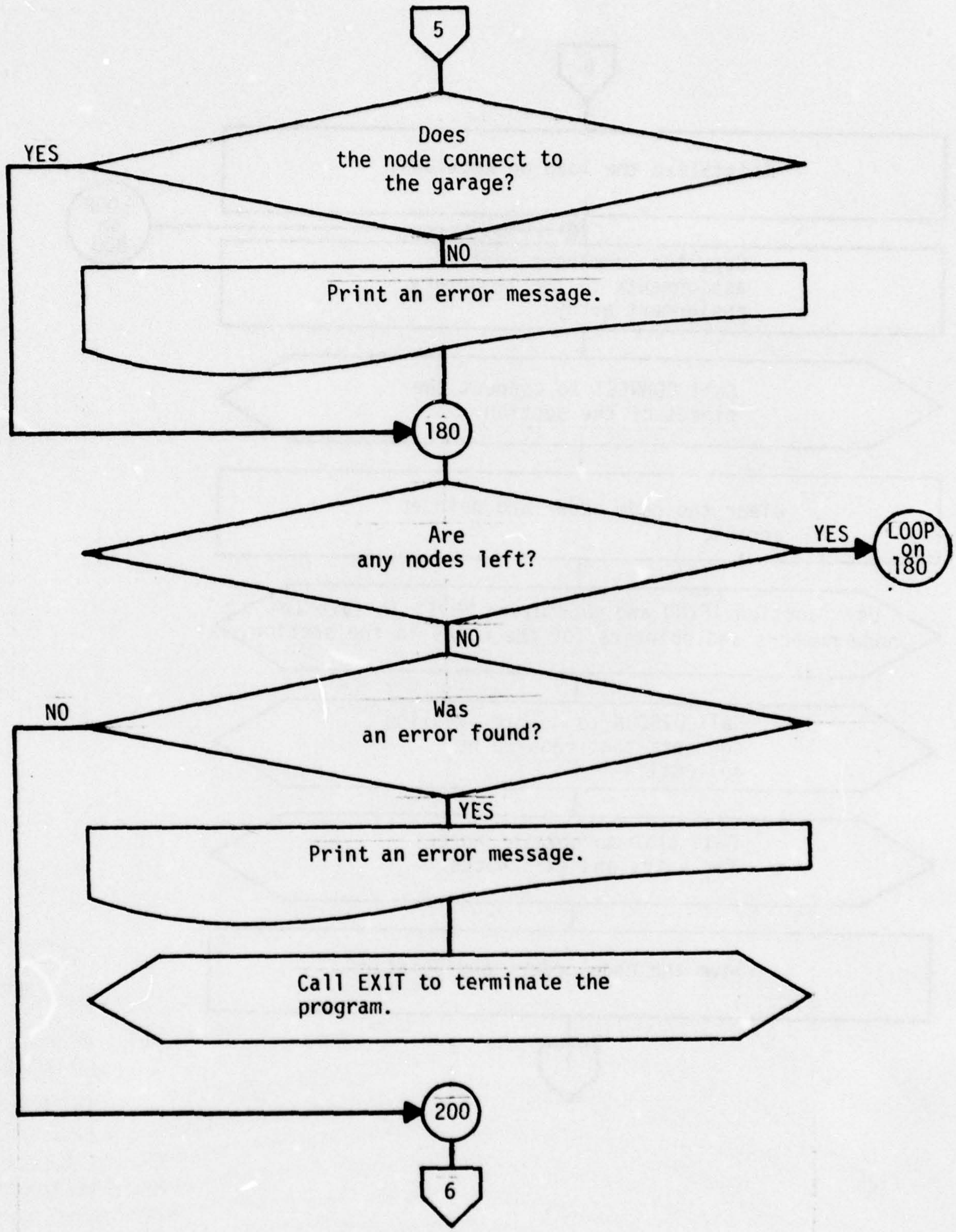
Program PHASE3



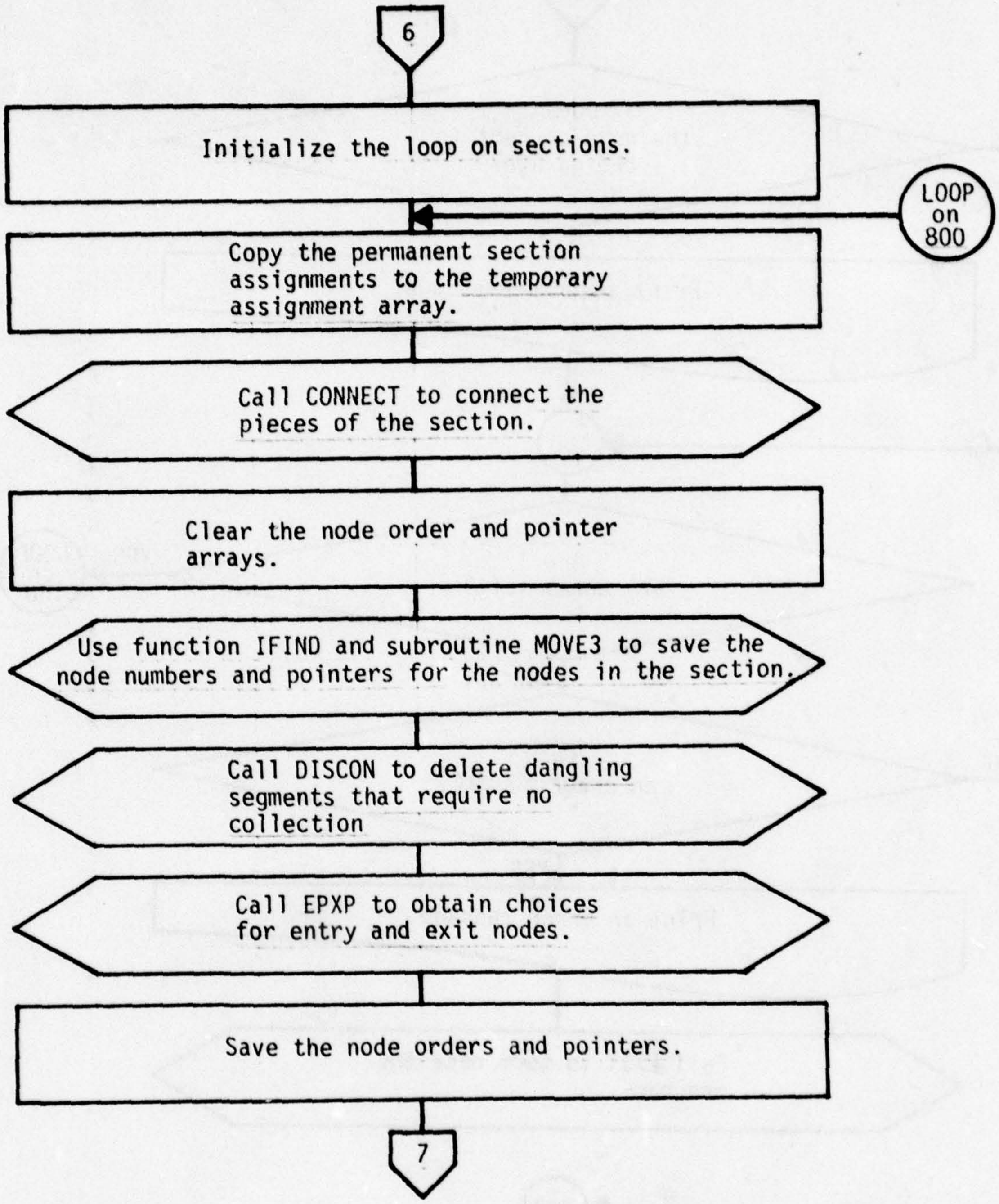
Program PHASE3



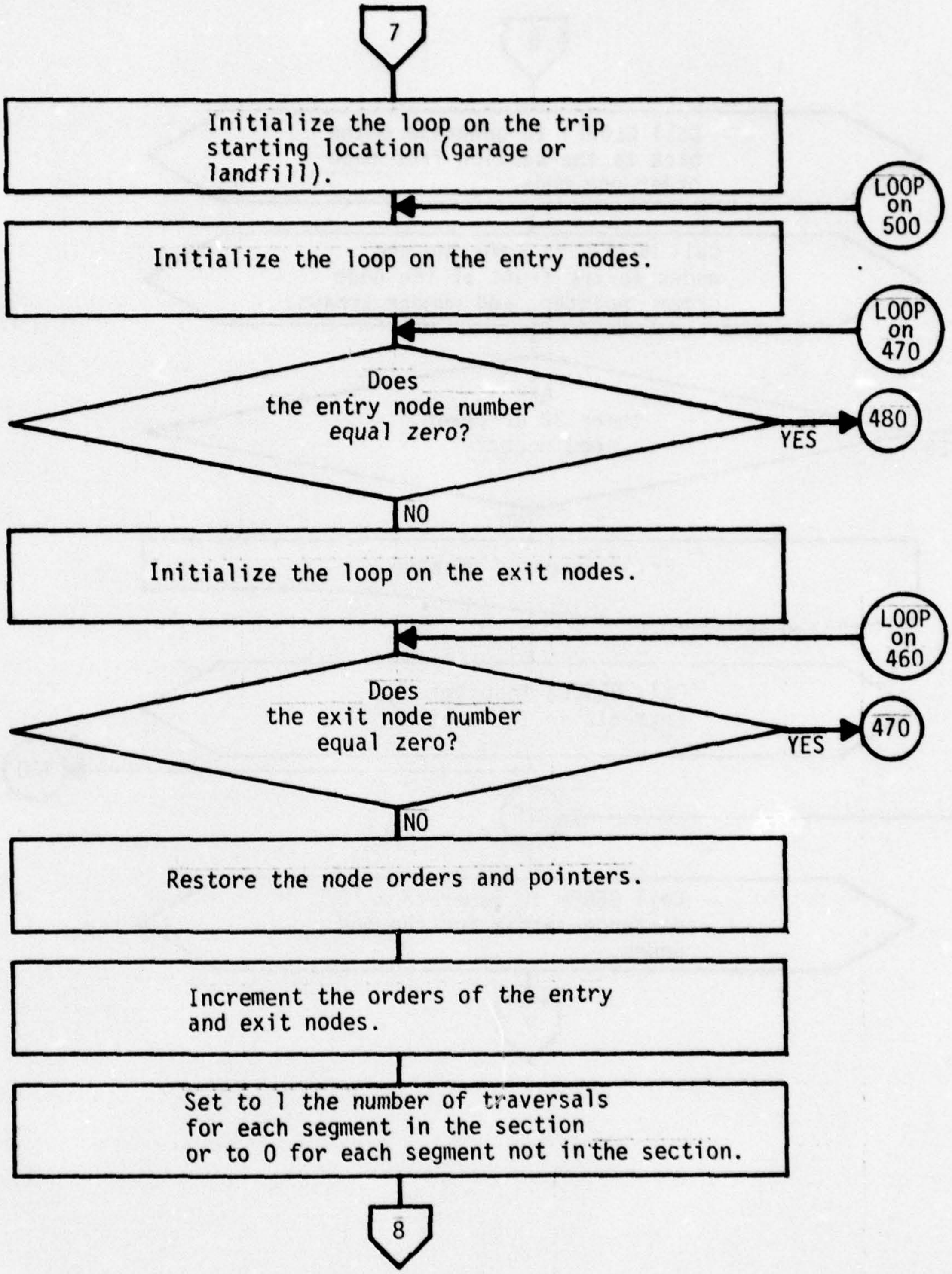
Program PHASE3



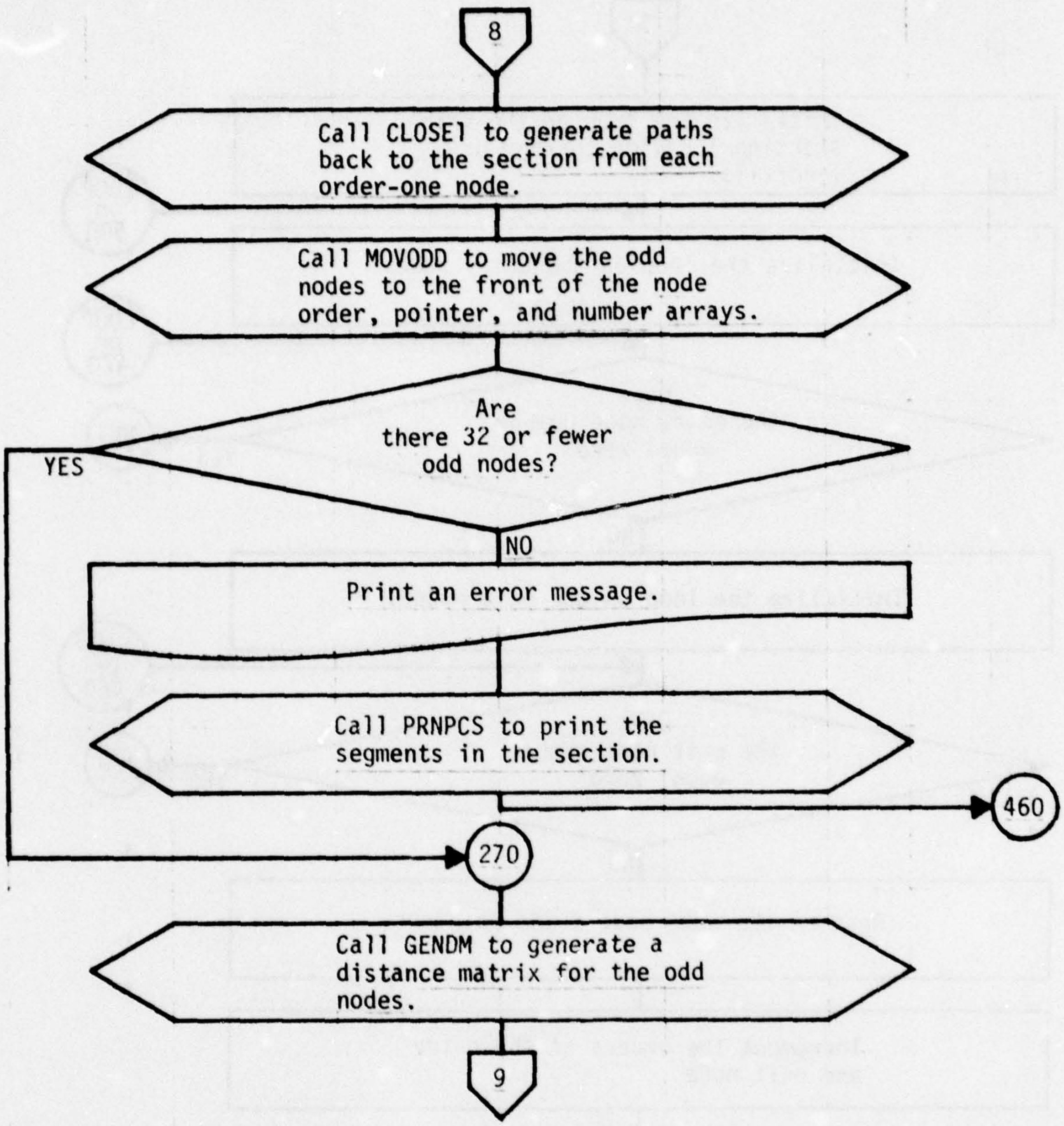
Program PHASE3



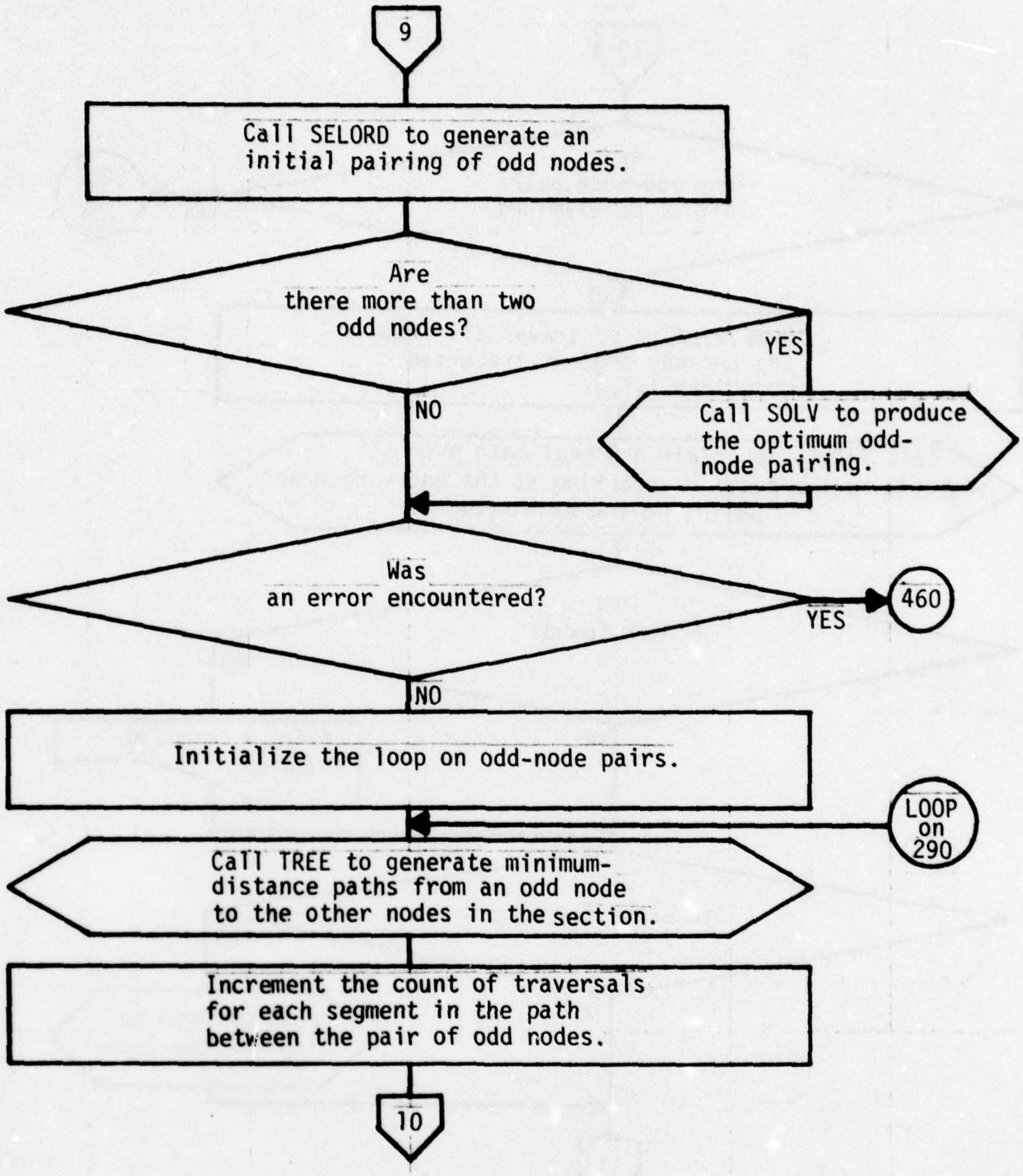
Program PHASE3



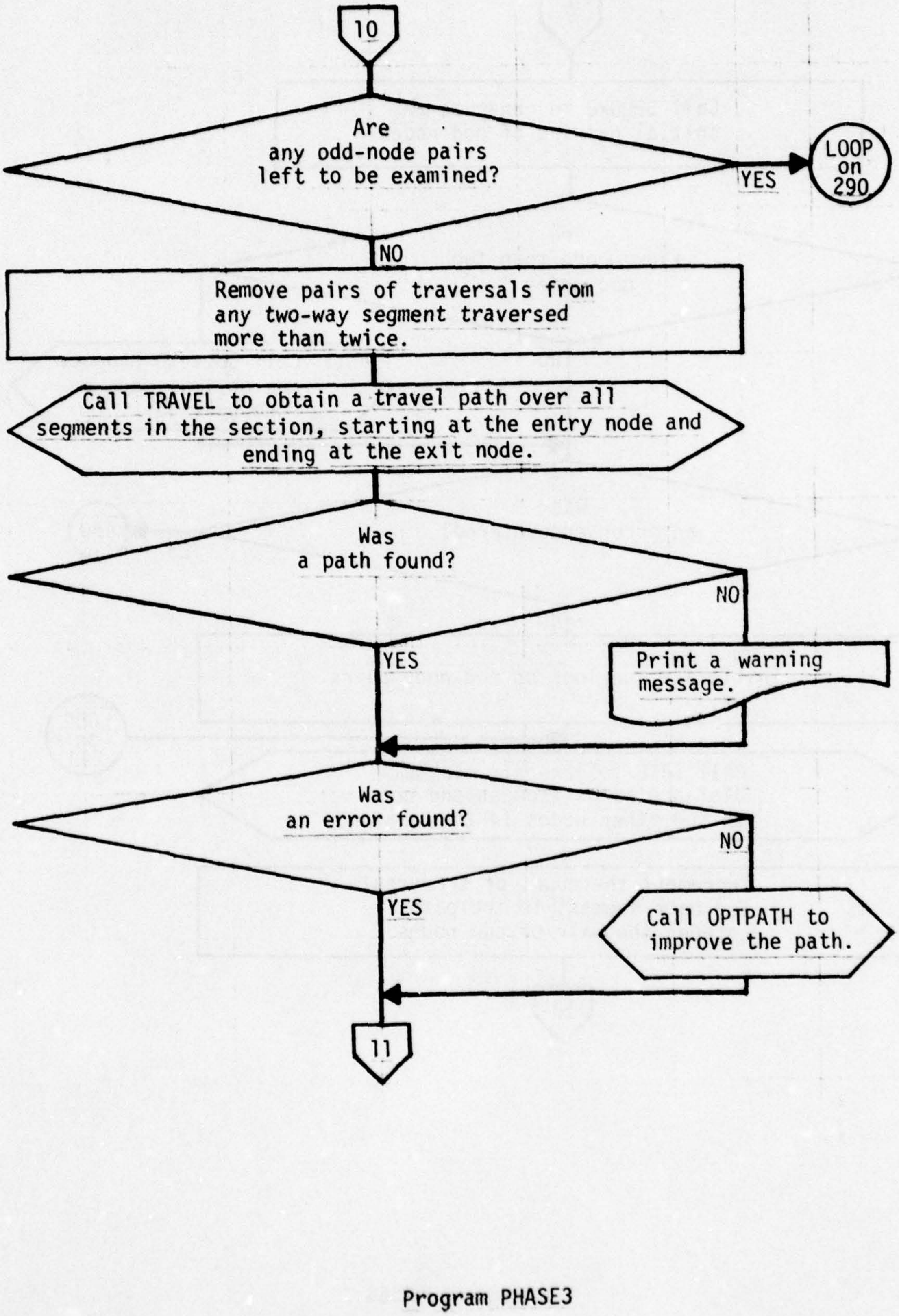
Program PHASE3



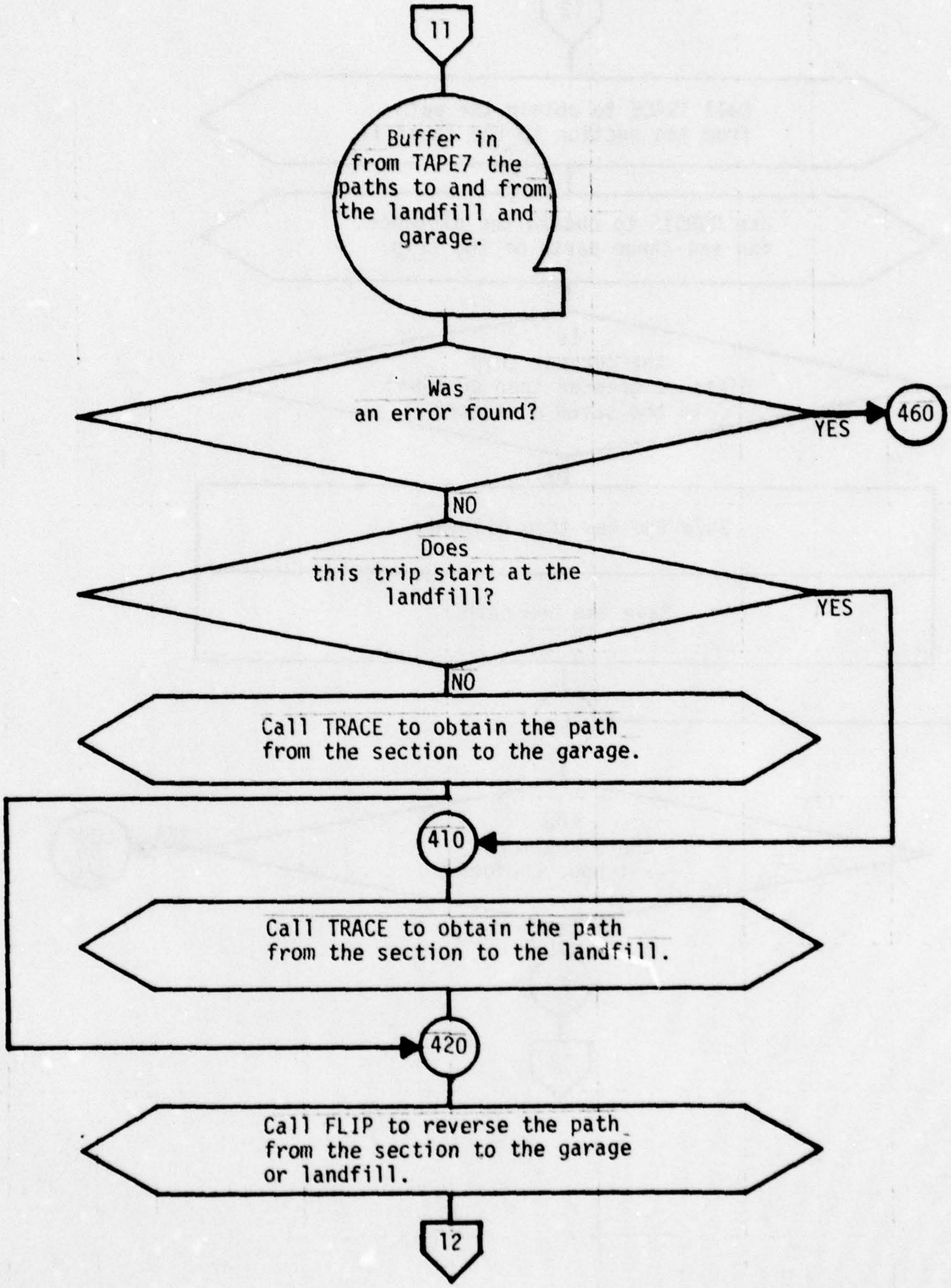
Program PHASE3



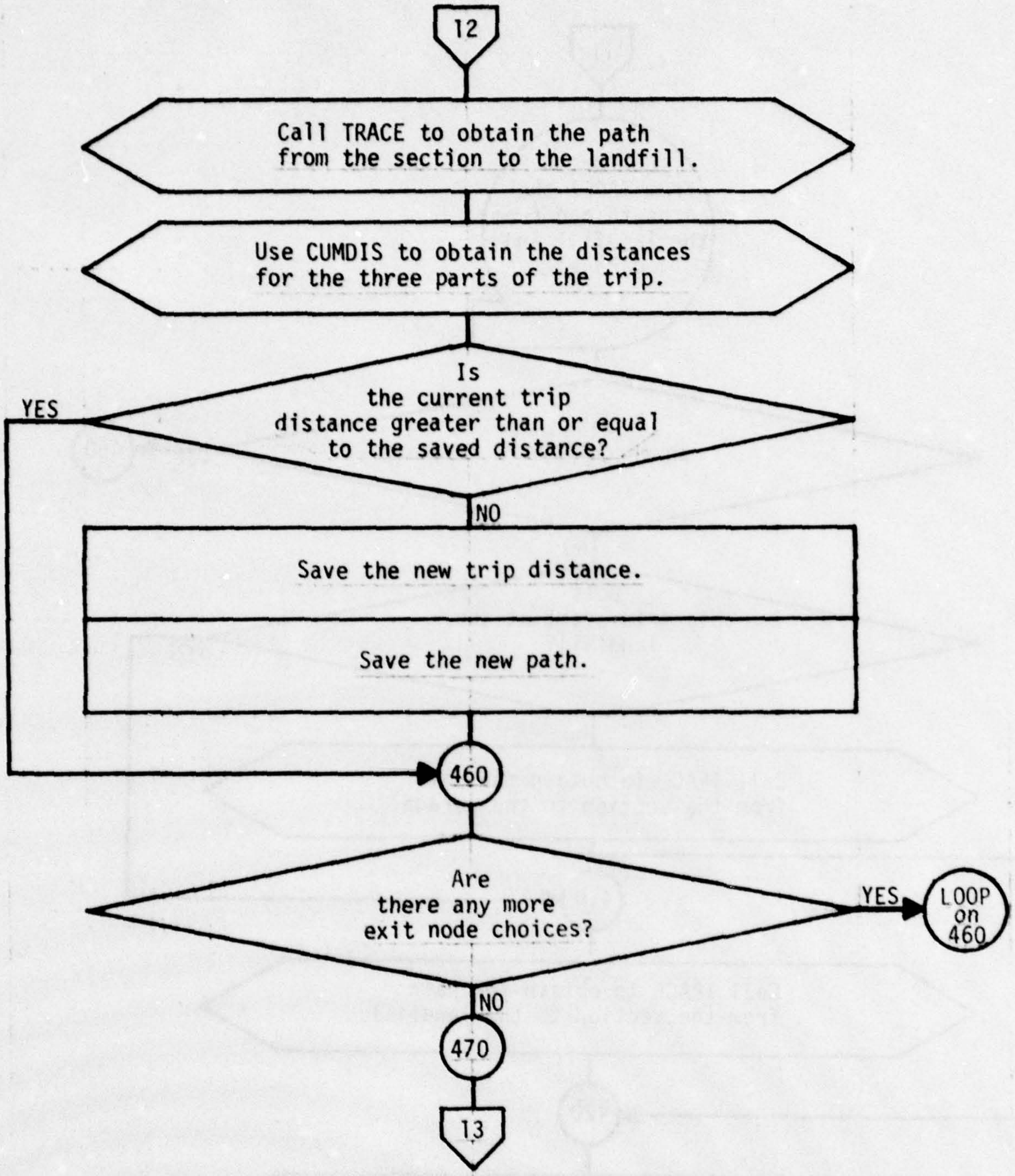
Program PHASE3



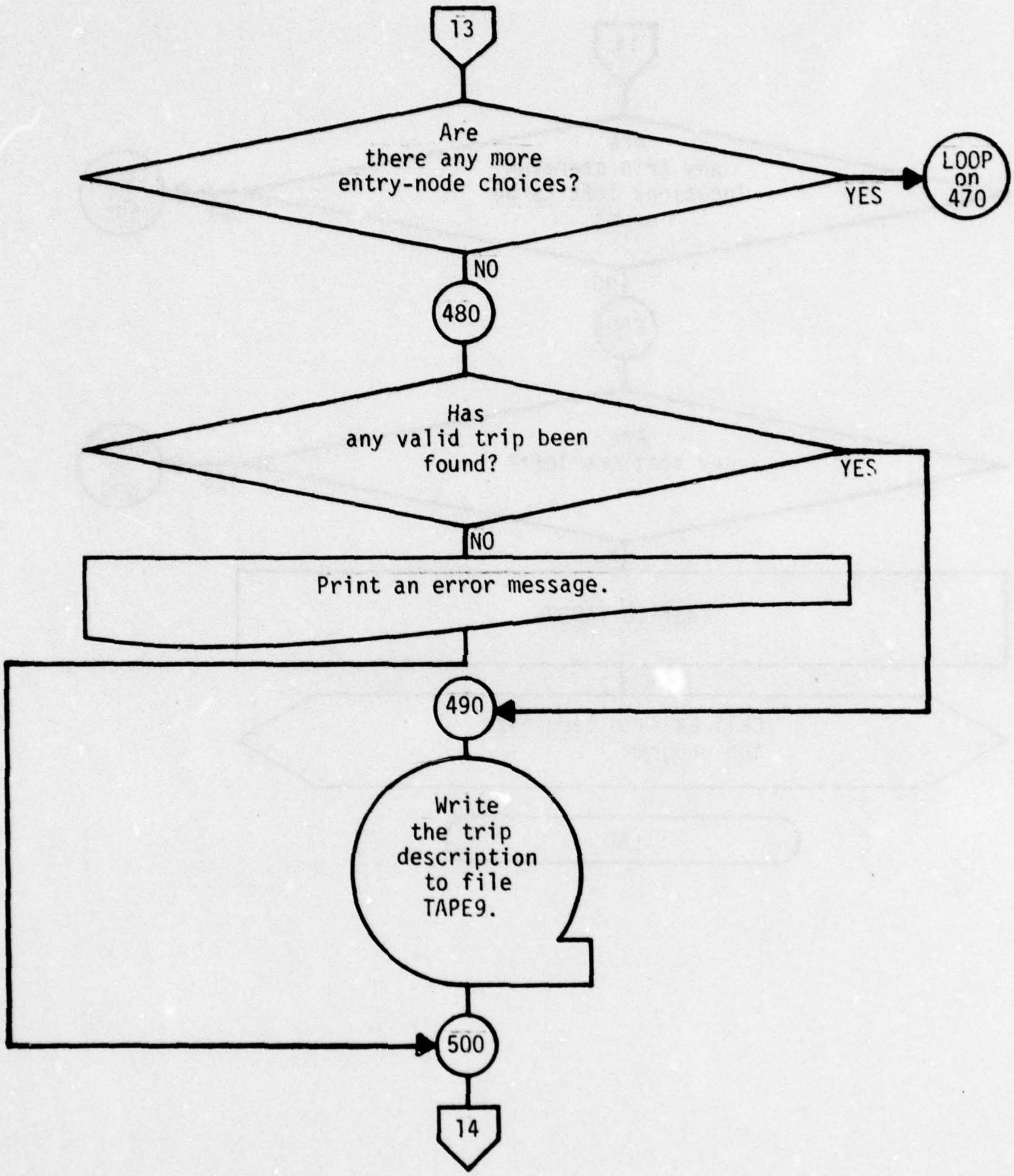
Program PHASE3



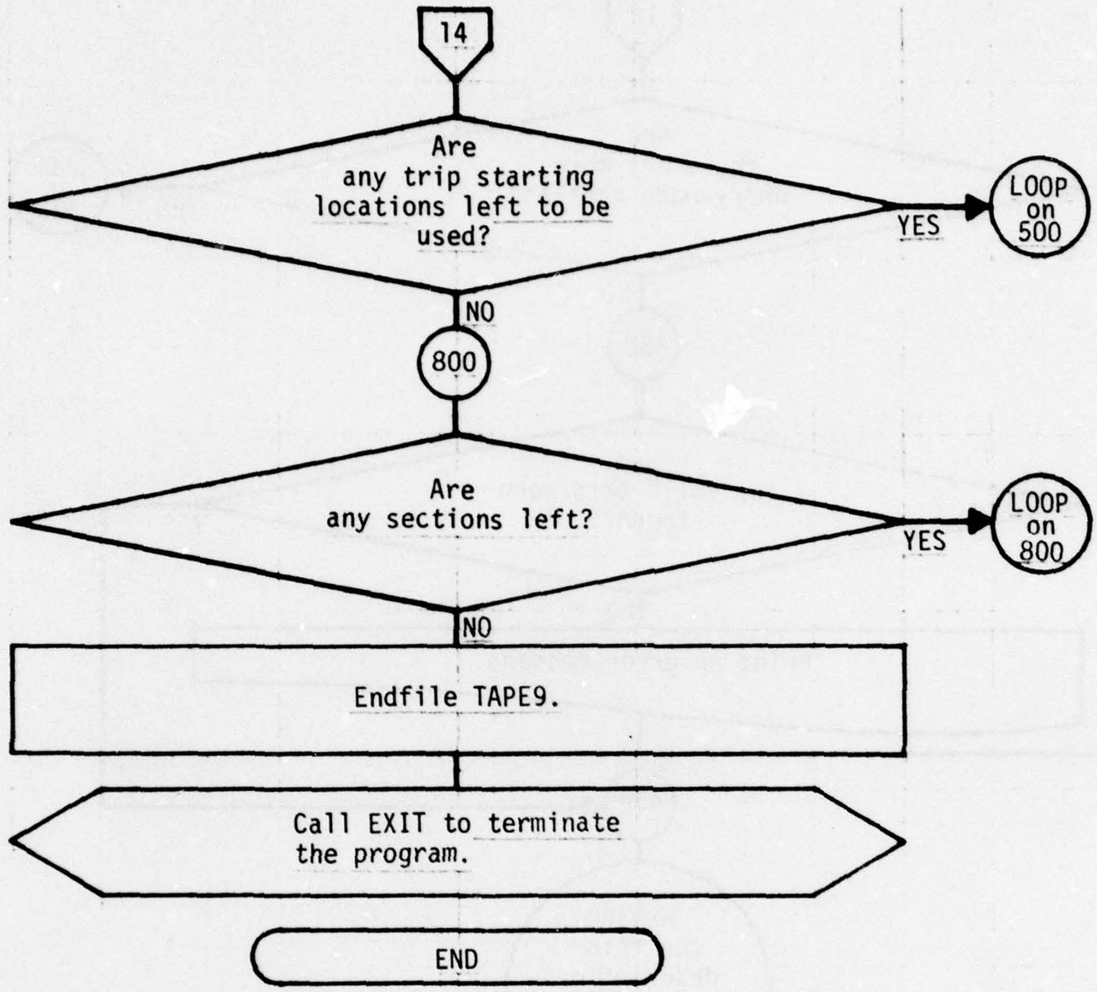
Program PHASE3



Program PHASE3



Program PHASE3



Program PHASE3

APPENDIX B

PROGRAM LISTINGS

```
      SUBROUTINE FLIF(X,N)                                              FLIP0010
      DIMENSION X(1)                                                    FLIP0020
      ND2=N/2                   $     NP1=N+1                           FLIP0030
      DO 10  I=1,ND2                                                    FLIP0040
      XT=X(I)                   $     X(I)=X(NP1-I)                     FLIP0050
   10 X(NP1-I)=XT                                                       FLIP0060
      RETURN                                                            FLIP0070
      END                                                               FLIP0080
```

```
      SUBROUTINE MOVE3(II,IF,A1,A2,A3)
C
C        LATEST CHANGES  --
C        NOV 18, 1975. HJI. ORIGINAL VERSION.
C
      DIMENSION A1(1), A2(1), A3(1)
      IF (IF .LT. II) GO TO 20
      DO 10 I=II,IF
      N=IF+II-I
      A1(N+1)=A1(N)              $     A2(N+1)=A2(N)
   10 A3(N+1)=A3(N)
   20 A1(II)=A2(II)=A3(II)=0.
      IF=IF+1
      RETURN
      END
```

```
      FUNCTION IFIND(NUM,IARRAY,LEN)                                    IFIND010
C                                                                       IFIND020
C        LATEST CHANGES --                                              IFIND030
C        APR 2, 1975.  HJI. MODIFIED TO RETURN NEGATIVE OF SUBSCRIPT    IFIND040
C        WHERE NUM SHOULD BE WHEN NUM IS NOT IN IARRAY.                 IFIND050
C        JAN 17, 1975. HJI. ORIGINAL VERSION.                           IFIND060
C                                                                       IFIND070
C        FUNCTION IFIND SEARCHES IARRAY(1)) THROUGH IARRAY(LEN) FOR NUM.IFIND080
C        IF NUM IS NOT FOUND, THE FUNCTION RETURNS WITH IFIND EQUAL     IFIND090
C        TO THE NEGATIVE OF THE SUBSCRIPT WHERE NUM SHOULD BE INSERTED. IFIND100
C        IF NUM=IARRAY(I), THE FUNCTION RETURNS WITH IFIND=I.           IFIND110
C                                                                       IFIND120
      DIMENSION IARRAY(1)                                               IFIND130
C                                                                       IFIND140
      IF (LEN .GT. 0) GO TO 5                                           IFIND150
      IFIND=-1                                                          IFIND160
      RETURN                                                            IFIND170
    5 II=1                                                              IFIND180
      IF=LEN                                                            IFIND190
   10 IP=(II+IF)/2                                                      IFIND200
      IF(NUM-IARRAY(IP)) 20,50,30                                       IFIND210
   20 IF=IP-1                                                           IFIND220
      GO TO 40                                                          IFIND230
   30 II=IP+1                                                           IFIND240
   40 IF (IF .GE. II) GO TO 10                                          IFIND250
      IFIND=-IP                                                         IFIND260
      IF (IARRAY(IP) .LT. NUM) IFIND=-1-IP                              IFIND270
      RETURN                                                            IFIND280
   50 IFIND=IP                                                          IFIND290
      RETURN                                                            IFIND300
      END                                                               IFIND310
```

```
      SUBROUTINE SHLSRT3(X,A,B,NW,SGN)                                  SHLS3010
      DIMENSION  X(1),  A(1),  B(1)                                     SHLS3020
C                                                                       SHLS3030
C        THIS SUBROUTINE SORTS NW WORDS STARTING AT   X(1).             SHLS3040
C        IF SGN = 1..  X IS SORTED IN INCREASING ORDER.                 SHLS3050
C        IF SGN =-1..  X IS SORTED IN DECREASING ORDER.                 SHLS3060
C        ARRAYS A AND B ARE REORDERED AS X IS SORTED SO THAT THEY       SHLS3070
C        ALWAYS CORRESPOND  TO THE SAME  X.                             SHLS3080
C                                                                       SHLS3090
      N=NW/2                                                            SHLS3100
   10 K=NW-N                                                            SHLS3110
      DO 50 I=1,K                                                       SHLS3120
      J=I                 $       L=I+N                                 SHLS3130
      XT=X(L)             $       AT=A(L)         $       BT=B(L)       SHLS3140
   20 IF (SGN*(XT-X(J)) .GE. 0.) GO TO 40                               SHLS3150
      X(L)=X(J)           $       A(L)=A(J)       $       B(L)=B(J)     SHLS3160
      L=J                 $       J=J-N           $       IF (J .GT. 0) GO TO 20SHLS3170
   40 X(L)=XT             $       A(L)=AT         $       B(L)=BT       SHLS3180
   50 CONTINUE                                                          SHLS3190
      IF (N .LE. 1) RETURN                                              SHLS3200
      N=(N+1)/2           $       GO TO 10                              SHLS3210
      END                                                               SHLS3220
```

```
      SUBROUTINE ISHLSRT(IX,IA,NW,ISGN,INC)                                 ISHLS010
                                                                            ISHLS020
C        LATEST CHANGES  --                                                 ISHLS030
C        NOV 19, 1975, HJI.  MAJOR REVISION --  ARGUMENT INC ADDED TO       ISHLS040
C        ALLOW SORTING WITH SPACING INC.                                    ISHLS050
                                                                            ISHLS060
      DIMENSION IX(INC,1),   IA(INC,1)                                      ISHLS070
C                                                                           ISHLS080
C        THIS SUBROUTINE SORTS NW WORDS IX(1),IX(1+INC),IX(1+2*INC),....ISHLS090
C        IF ISGN = 1, IX IS SORTED IN INCREASING ORDER.                     ISHLS100
C        IF ISGN =-1, IX IS SORTED IN DECREASING ORDER.                     ISHLS110
C        ARRAY IA IS REORDERED AS IX IS SORTED SO THAT EACH IA              ISHLS120
C        ALWAYS CORRESPONDS TO THE SAME IX.                                 ISHLS130
C                                                                           ISHLS140
      N=NW/2                                                                ISHLS150
   10 K=NW-N                                                                ISHLS160
      DO 50 I=1,K                                                           ISHLS170
      J=I                                                                   ISHLS180
      L=I+N                                                                 ISHLS190
      IXT=IX(1,L)                 $       IAT=IA(1,L)                       ISHLS200
   20 IF (ISGN*(IXT-IX(1,J))) 30,40,40                                      ISHLS210
   30 IX(1,L)=IX(1,J)             $       IA(1,L)=IA(1,J)                   ISHLS220
      L=J                                                                   ISHLS230
      J=J-N                                                                 ISHLS240
      IF (J .GT. 0)  GO TO 20                                               ISHLS250
   40 IX(1,L)=IXT                 $       IA(1,L)=IAT                       ISHLS260
   50 CONTINUE                                                              ISHLS270
      IF (N .LE. 1)  RETURN                                                 ISHLS280
      N=(N+1)/2                                                             ISHLS290
      GO TO 10                                                              ISHLS300
      END                                                                   ISHLS310
```

```
      SUBROUTINE ADJUST(TL,TC)
C
C         LATEST CHANGES  --
C         JULY 7, 1976. HJI.   ORIGINAL VERSION.
C
C         THIS SUBROUTINE EXAMINES PAIRS OF SEGMENTS AT ORDER 2 NODES AND
C         PUTS THEM IN THE SAME SECTION IF POSSIBLE.
C
      COMMON NSEG,STG(11,500)
      COMMON /NODDATA/ KNODES,NODNUM(300),XNOD(300),YNOD(300),
     1  NUMNBR(300),NBRSEG(300)
      DIMENSION ISTG(11,500), TC(50),TL(50)
      EQUIVALENCE (ISTG,STG)
      DATA NH,NROF,NSECT/ 5,8,9/
C
      DO 70 I=1,KNODES
      IF (NUMNBR(I) .NE. 2) GO TO 70
      NSG1=NBRSEG(I) .A. 1777B     $     NSG2=SHIFT(NBRSEG(I),-10)
      NSC1=ISTG(NSECT,NSG1)        $    NSC2=ISTG(NSECT,NSG2)
      IF (NSC1.EQ.NSC2 .OR. NSC1.EQ.0 .OR. NSC2.EQ.0) GO TO 70
      R1=ISTG(NH,NSG1)*STG(NROF,NSG1)
      R2=ISTG(NH,NSG2)*STG(NROF,NSG2)
      IF (R1 .EQ. 0. .AND. R2 .EQ. 0.) GO TO 70
      IF (R2 .LE. R1) GO TO 30
      IF (TL(NSC2)+R1 .GT. TC(NSC2)) GO TO 50
   10 TL(NSC2)=TL(NSC2)+R1    $      TL(NSC1)=TL(NSC1)-R1
      ISTG(NSECT,NSG1)=NSC2  $      PRINT 20, NSG1,NSC1,NSC2,R1
   20 FORMAT(3I9,F9.2,*    (PROGRAM ADJUSTED)*)
      GO TO 70
   30 IF (TL(NSC1)+R2 .GT. TC(NSC1)) GO TO 60
   40 TL(NSC1)=TL(NSC1)+R2    $      TL(NSC2)=TL(NSC2)-R2
      ISTG(NSECT,NSG2)=NSC1  $      PRINT 20, NSG2,NSC2,NSC1,R2
      GO TO 70
   50 IF (TL(NSC1)+R2 .GT. TC(NSC1)) 70,40
   60 IF (TL(NSC2)+R1 .LE. TC(NSC2)) GO TO 10
   70 CONTINUE
      RETURN
      END
```

```
      FUNCTION CUMDIS(ISEG,NSG)
C       LATEST CHANGES --
C       JAN 4, 1977. HJI.    ORIGINAL VERSION

      COMMON NSEG,ISTG(11,500)
      DIMENSION ISEG(1),STG(11,500)
      EQUIVALENCE (STG,ISTG)
      DATA NSTR,LEN/1,4/

      SUM=0.
      DO 10 I=1,NSG
      J=ISEG(I)
      IF (J .GT. NSEG) ISEG(I)=ISTG(NSTR,J)
   10 SUM=SUM+STG(LEN,ISEG(I))
      CUMDIS=SUM
      RETURN
      END
```

```
      SUBROUTINE PRNFCS(NTRIP,NPIECE)                                   PRPCS010
      COMMON NSEG,ISTG(11,300)                                          PRPCS020
      DATA MSECT/10/                                                    PRPCS030
                                                                        PRPCS040
      IZERO=0                                                           PRPCS050
      DO 40 I=1,NPIECE                                                  PRPCS060
      ITRIP=NTRIP+1000*(I-1)       $     CC=1H     $     NN=0           PRPCS070
      PRINT 10, I                                                       PRPCS080
   10 FORMAT(*0SEGMENTS IN PIECE *,I3/)                                 PRPCS090
         DO 30 J=1,NSEG                                                 PRPCS100
         IF (ISTG(MSECT,J) .NE. ITRIP) GO TO 30                         PRPCS110
         NN=NN+1                                                        PRPCS120
         PRINT 20, CC,(IZERO,K=1,NN),J                                  PRPCS130
   20    FORMAT(A1,31I4.0)                                              PRPCS140
         CC=1H+              $     IF (NN .LT. 30) GO TO 30             PRPCS150
         CC=1H               $     NN=0                                 PRPCS160
   30    CONTINUE                                                       PRPCS170
   40 CONTINUE                                                          PRPCS180
      RETURN                                                            PRPCS190
      END                                                               PRPCS200
```

```
      SUBROUTINE TRACE(NSTART,NSTOP,LPREVN,NPREVS,IPS,IPN,NSG)

C         LATEST CHANGES --
C         DEC 21, 1976. HJI.   ORIGINAL VERSION.
C         JAN 4. 1977.  HJI    ADDED IPN ARRAY.

      COMMON /NODDATA/KNODES,NODNUM(300)
      DIMENSION LPREVN(1),NPREVS(1),IPN(1),IPS(1)

      L=IFIND(NSTART,NODNUM,KNODES)      $      IPN(1)=NSTART
      LEND=IFIND(NSTOP,NODNUM,KNODES)
      NSG=0

   10 NSG=NSG+1
      IPS(NSG)=NPREVS(L)
      L=LPREVN(L)                 $      IPN(NSG+1)=NODNUM(L)
      IF (L .LT. 0   .OR.  L .GT. KNODES  .OR.  NSG .GT. 200) GO TO 20
      IF (L .NE. LEND .AND. L .NE. 0) GO TO 10
      RETURN
   20 PRINT 30,NSTART,NSTOP,NSG,L,KNODES,(I,NODNUM(I),LPREVN(I),
     1  NPREVS(I),I=1,KNODES)
   30 FORMAT(/*0IMPROPER LINE POINTER USED IN TRACE*/*0JOB TERMINATED*/
     1  *0NSTART=*,I5,*     NSTOP=*,I5,*     NSG=*,I5,*     L=*,I5,
     2  *     KNODES=*,I5/*0 LINE     NODE   LPREVN  NPREVS*/ (1X,I5,3I8))
      CALL DUMP(0,1170008,4)
      END
```

```
      SUBROUTINE MOVODD(KN,NODORD,NODPTR,NODD)                          MOVOD010
                                                                        MOVOD020
C        LATEST CHANGES --                                              MOVOD030
C        JULY 28, 1976. HJI. ORIGINAL VERSION.                          MOVOD040
                                                                        MOVOD050
C        MOVODD COUNTS AND MOVES ODD ORDER NODES TO THE FRONT OF ARRAYS MOVOD060
C        NODORD AND NODPTR.                                             MOVOD070
      DIMENSION NODORD(100),NODPTR(100)                                 MOVOD080
                                                                        MOVOD090
      JJ=0          $     KK=KN+1                                       MOVOD100
C        FIND AN EVEN ORDER NODE                                        MOVOD110
   10 JJ=JJ+1       $     IF (JJ .GE. KK) GO TO 30                      MOVOD120
      IF (MOD(NODORD(JJ),2) .EQ. 1) GO TO 10                            MOVOD130
C        FIND AN ODD ORDER NODE                                         MOVOD140
   20 KK=KK-1       $     IF (JJ .GE. KK) GO TO 30                      MOVOD150
      IF (MOD(NODORD(KK),2) .EQ. 0) GO TO 20                            MOVOD160
C        INTERCHANGE                                                    MOVOD170
      N=NODORD(KK) $     NODORD(KK)=NODORD(JJ)     $     NODORD(JJ)=N   MOVOD180
      N=NODPTR(KK) $     NODPTR(KK)=NODPTR(JJ)     $     NODPTR(JJ)=N   MOVOD190
      GO TO 10                                                          MOVOD200
   30 NODD=JJ-1                                                         MOVOD210
      RETURN                                                            MOVOD220
      END                                                               MOVOD230
```

```
      SUBROUTINE TREE(NORG,DIST,TIME,W,NPREVN,NPREVS,IDIR,MAXLVL,INSECT)

C        LATEST CHANGES  --
C        JULY 16, 1976. HJI. MAJOR REVISION --  ARGUMENTS TIME AND W
C        ADDED.
C        JUNE 3, 1976. HJI.  MAJOR REVISION --  ARGUMENTS NPREVS AND
C        INSECT ADDED.
C        OCT 15, 1975. HJI. MAJOR REVISION  --  ARGUMENT MAXLVL ADDED
C        TO STOP TREE BUILDING AFTER MAXLVL LEVELS OF NODES.
C        OCT 9, 1975. HJI.   ADDED 3 STATEMENTS JUST BEFORE SN730 TO
C        PATCH PROBLEM CAUSED WHEN THE LAST NODE ADDED IS DISCONNECTED
C        (TO BE REPLACED BY ITSELF).
C        OCT 3, 1975. HJI.   NSEG PUT IN BLANK COMMON.   900 LOOP
C        CHANGED TO SCAN ONLY NEIGHBORING SEGMENTS RATHER THAN ENTIRE
C        SEGMENT LIST.

C     NORG   = STARTING NODE NUMBER
C     DIST   = ARRAY OF DISTANCES (MILES) FROM NORG
C     TIME   = ARRAY OF TIMES (MINUTES) FROM NORG
C     W      = 0. TO MINIMIZE TIME,  1. TO MINIMIZE DISTANCE
C     NPREVN= ARRAY OF PREVIOUS NODE LINE NUMBERS
C     NPREVS= ARRAY OF PREVIOUS SEGMENT LINE NUMBERS
C     IDIR   =  1 IF LEAVING NORG, -1 IF GOING TO NORG
C     MAXLVL= MAXIMUM LEVELS IN TREE
C     INSECT=  0 IF ALL SEGMENTS MAY BE USED, ELSE ONLY SEGMENTS IN
C              SECTION INSECT MAY BE USED.

      REAL NIU,LEN
      COMMON NSEG,STG
      COMMON /NODDATA/ KNODES,NODNUM(300),XNOD(300),YNOD(300),
     1  NUMNBR(300),NBRSEG(300)
      DIMENSION DIST(400),IST(400,4),          NIU(400),STG(11,500),
     1  ISTG(11,500),NPREVN(400),NPREVS(400), NIULOC(400), TIME(400)
      EQUIVALENCE (ISTG,STG), (KNODES,MAXNOD)
      DATA NN1,NN2,NLEN,NSPD,NWY,MSECT/ 2,3,4,6,7,10/

C        FUNCTIONS
      INS(II)=ISTG(MSECT,II)
      NU1(II)=ISTG(NN1,II)
```

```
      NU2(II)=ISTG(NN2,II)                                              TREE0400
      LEN(II)=STG(NLEN,II)                                              TREE0410
      TIM(II)=60.*STG(NLEN,II)/STG(NSPD,II)                             TREE0420
      NWAY(II)=ISTG(NWY,II)                                             TREE0430
                                                                        TREE0440
      CW=1.-W                                                           TREE0450
      I=IFIND(NORG,NODNUM,KNODES)                                       TREE0460
      IF (I .GT. 0) GO TO 570                                           TREE0470
      PRINT 550, NORG                                                   TREE0480
  550 FORMAT (*0NODE*,I5,* IS NOT IN THE NODE TABLE*/* CORRECT THE DUMP OTREE0490
     1R GARAGE NUMBER*/* JOB TERMINATED*)                               TREE0500
      CALL EXIT                                                         TREE0510
  570 CONTINUE                                                          TREE0520
C                                                                       TREE0530
      DO 600 J=1,MAXNOD                                                 TREE0540
      DO 580 K=1,4                                                      TREE0550
  580 IST(J,K)=0                                                        TREE0560
      NIU(J)=0                                                          TREE0570
      NIULOC(J)=0                                                       TREE0580
  600 IST(J,4)=J+1                                                      TREE0590
C                                                                       TREE0600
      NEXT=IST(1,4)                                                     TREE0610
      IST(1,1)=I                                                        TREE0620
      IST(1,2)=0                                                        TREE0630
      IST(1,3)=0                                                        TREE0640
      IST(1,4)=0                                                        TREE0650
      CUM=0                                                             TREE0660
      NIU(I)=-1000                                                      TREE0670
      NMRT=1                                                            TREE0680
C                                                                       TREE0690
C   MIRROR POSTORDER TREE TRAVERSAL (RT, ROOT, LEFT)                    TREE0700
C                                                                       TREE0710
      DO 940 J=1,MAXLVL                                                 TREE0720
      NEWS=0                                                            TREE0730
      K=1                                                               TREE0740
C                                                                       TREE0750
  620 NEXTK=IST(K,3)                                                    TREE0760
      IF (NEXTK .EQ. 0) GO TO 640                                       TREE0770
      NMRT=NMRT+1                                                       TREE0780
```

```
      K=NEXTK
      GO TO 620
  640 IF (NMRT .EQ. J) GO TO 680
  660 NEXTK=IST(K,2)
      K=IABS(NEXTK)
      IF (NEXTK) 670,930,620
  670 NMRT=NMRT-1
      GO TO 660
  680 IFIRST=1
      N=IST(K,1)
      LAST=K
C
C        SEEK A NODE WHICH CONNECTS TO NODE NODNUM(N)
C
      KSEG=NUMNBR(N)
        DO 900 KKK=1,KSEG
        KK=SHIFT(NBRSEG(N),10*(1-KKK)) .A. 17778
         IF (INSECT .GT. 0   .AND. INSECT .NE. INS(KK)) GO TO 900
        L=NU2(KK)
        IF (NU1(KK) .EQ. NODNUM(N) .AND. (IDIR .GT. 0 .OR. NWAY(KK) .EQ.
     1  2)) GO TO 700
        IF (L .NE. NODNUM(N) .OR. (IDIR .GT. 0 .AND. NWAY(KK) .EQ. 1))
     1  GO TO 900
        L=NU1(KK)
  700   L=IFIND(L,NODNUM,KNODES)
        CUM=W*(LEN(KK)+DIST(N))+CW*(TIM(KK)+TIME(N))
        IF (NIU(L)) 900,840,720
  720   IF (CUM .GE. NIU(L)) GO TO 900
C
C    DISCONNECT OLD ENTRY NODE NODNUM(L)
C
        LB=NIULOC(L)
        LA=IST(LB,4)
        LC=IST(LB,2)
        IF (LB .NE. LAST) GO TO 730
        LAST=LA
        IF (LA+LC .EQ. 0) IFIRST=1
  730   IF (IST(LA,3) .EQ. LB) GO TO 740
        IST(LA,2)=LC
```

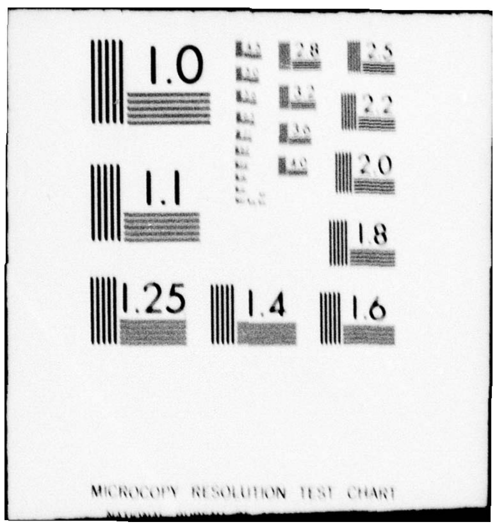
```
      GO TO 760                                                         TREE1180
  740 IST(LA,3)=MAX0(0,LC)                                              TREE1190
  760 IF (LC .GT. 0) IST(LC,4)=IST(LB,4)                                TREE1200
      IST(LB,2)=0                                                       TREE1210
      LC=LB                                                             TREE1220
C                                                                       TREE1230
C   REMOVE TREE STARTING AT OLD NODE NODNUM(L)                          TREE1240
  780 LB=LC                                                             TREE1250
  800 LC=IST(LB,3)                                                      TREE1260
      IF (LC .GT. 0) GO TO 780                                          TREE1270
  820 NA=IST(LB,1)                                                      TREE1280
      NIU(NA)=0                                                         TREE1290
      NIULOC(NA)=0                                                      TREE1300
      IST(LB,4)=NEXT                                                    TREE1310
      NEXT=LB                                                           TREE1320
      LC=IST(LB,2)                                                      TREE1330
      LB=IABS(LC)                                                       TREE1340
      IF (LC) 820,840,800                                               TREE1350
  840 CONTINUE                                                          TREE1360
C                                                                       TREE1370
C    ADD A NEW NODE NODNUM(L)                                           TREE1380
C                                                                       TREE1390
      NEWS=1                                                            TREE1400
      IST(NEXT,1)=L                                                     TREE1410
      IST(NEXT,2)=IST(LAST,2)                                           TREE1420
      NPREVN(L)=IST(-IST(LAST,2),1)                                     TREE1430
      NPREVS(L)=KK                                                      TREE1440
      IST(NEXT,3)=0                                                     TREE1450
      NEWNXT=IST(NEXT,4)                                                TREE1460
      IST(NEXT,4)=LAST                                                  TREE1470
      DIST(L)=DIST(N)+LEN(KK)    $     TIME(L)=TIME(N)+TIM(KK)          TREE1480
      NIU(L)=CUM                                                        TREE1490
      NIULOC(L)=NEXT                                                    TREE1500
      IF (IFIRST .EQ. 1) GO TO 860                                      TREE1510
      IST(LAST,2)=NEXT                                                  TREE1520
      GO TO 880                                                         TREE1530
  860 IST(LAST,3)=NEXT                                                  TREE1540
      IST(NEXT,2)=-LAST                                                 TREE1550
      NPREVN(L)=IST(LAST,1)                                             TREE1560
```

```
      IFIRST=0
 880  LAST=NEXT
      NEXT=NEWNXT
 900  CONTINUE
C
     IF (IFIRST .EQ. 0) K=-IST(LAST,2)
 920 K=IST(K,2)
     IF (K .GT. 0) GO TO 620
     NMRT=NMRT-1
     K=-K
     IF (K .GT. 0) GO TO 920
     NMRT=NMRT+1
 930 CONTINUE
C
     IF (NEWS .EQ. 0) GO TO 960
C
 940 CONTINUE
 960 CONTINUE
     RETURN
     END
```

AD-A060 986 NEW MEXICO UNIV ALBUQUERQUE ERIC H WANG CIVIL ENGINE--ETC F/G 13/2
AIR FORCE REFUSE-COLLECTION SCHEDULING PROGRAM DESCRIPTION. VOL--ETC(U)
JUN 78 H J IUZZOLINO F29601-76-C-0015
UNCLASSIFIED CERF-EE-21 CEEDO-TR-78-23-VOL-3 NL

3 OF 3
AD
A060 986

END
DATE
FILMED
2-79
DDC


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART

```
      SUBROUTINE CON2ST(NTRIP,NCON)

C        LATEST CHANGES  --
C        JUNE 17, 1976. HJI. ORIGINAL VERSION

C        THIS SUBROUTINE CONNECTS PIECES OF A SECTION BY TWO STEP PATHS

      COMMON NSEG,STG
      COMMON /NODDATA/ KNODES,NODNUM(300),XNOD(300),YNOD(300),
     1  NUMNBR(300),NBRSEG(300)
      COMMON /TEMSTG/ NP1(300),NP2(300),NS1(300),NS2(300),SLEN(300)
      DIMENSION ISTG(11,500), STG(11,500)
      EQUIVALENCE (STG,ISTG)
      DATA NN1,NN2,LEN,NWAY,MSECT/ 2,3,4,7,10/

      NS=0
      DO 10 I=1,300
   10 NP1(I)=NP2(I)=NS1(I)=NS2(I)=SLEN(I)=0.
      DO 140 LINE=1,KNODES
      NODE=NODNUM(LINE)
      LIM=NUMNBR(LINE)        $      IF (LIM .EQ. 1) GO TO 140
      LIMM1=LIM-1             $      NBRS1=NBRSEG(LINE)
         DO 130 I=1,LIMM1
         ISEG=SHIFT(NBRS1,10*(1-I)) .A. 1777B
         IF (MOD(ISTG(MSECT,ISEG),1000) .EQ. NTRIP) GO TO 140
         K=ISTG(NN1,ISEG)     $      IF (K .EQ. NODE) K=ISTG(NN2,ISEG)
         L=IFIND(K,NODNUM,KNODES)
         LIM2=NUMNBR(L)       $      NBRS2=NBRSEG(L)
C        KEEP ISEG ONLY IF SOME SEGMENT BORDERING NODE K IS IN A PIECE
C        OF SECTION NTRIP.
            DO 20 L=1,LIM2
            KSEG=SHIFT(NBRS2,10*(1-L)) .A. 1777B
            IF (KSEG .EQ. ISEG) GO TO 20
            JSECT=ISTG(MSECT,KSEG)
            IF (MOD(JSECT,1000) .EQ. NTRIP) GO TO 30
   20       CONTINUE
         GO TO 130
C        SEARCH FOR ANOTHER SEGMENT BORDERING NODE WHICH ENDS AT A PIECE
C        OF SECTION NTRIP.
```

```
 30     IP1=I+1                                                          CN2S0400
          DO 120 J=IP1,LIM                                               CN2S0410
          JSEG=SHIFT(NBRS1,10*(1-J)) .A. 1777B                           CN2S0420
          IF (MOD(ISTG(MSECT,JSEG),1000) .EQ. NTRIP) GO TC 140           CN2S0430
          K=ISTG(NN1,JSEG) $      IF (K .EQ. NODE) K=ISTG(NN2,JSEG)      CN2S0440
          L=IFIND(K,NODNUM,KNODES)                                       CN2S0450
          LIM2=NUMNBR(L)   $      NBRS2=NBRSEG(L)                        CN2S0460
C       KEEP JSEG ONLY IF SOME SEGMENT BORDERING NODE K IS IN A PIECE    CN2S0470
C       OF SECTION NTRIP                                                 CN2S0480
            DO 40 L=1,LIM2                                               CN2S0490
            KSEG=SHIFT(NBRS2,10*(1-L)) .A. 1777B                         CN2S0500
            IF (KSEG .EQ. JSEG) GC TO 40                                 CN2S0510
            KSECT=ISTG(MSECT,KSEG)                                       CN2S0520
            IF (MOD(KSECT,1000) .EQ. NTRIP) GO TC 50                     CN2S0530
 40         CONTINUE                                                     CN2S0540
          GO TO 120                                                      CN2S0550
 50       IF (JSECT-KSECT) 70,120,60                                     CN2S0560
C       SEGMENTS ISEG AND JSEG TOGETHER CONNECT TWO DISJOINT PIECES OF   CN2S0570
C       SECTION NTRIP.  SAVE THE CONNECTION IF THERE IS NO SHORTER ONE.CN2S0580
                                                                         CN2S0590
C       PUT THE SMALLER PIECE NUMBER IN JSECT                            CN2S0600
 60       L=JSECT          $      JSECT=KSECT    $      KSECT=L          CN2S0610
C       CHECK FOR TWO ONE WAY SEGMENTS POINTING AT EACH OTHER            CN2S0620
 70       IF (ISTG(NWAY,ISEG) .NE. 1  .OR.  ISTG(NWAY,JSEG) .NE. 1)      CN2S0630
C       CHECK FOP TWC ONE WAY SEGMENTS POINTING AT EACH OTHER            CN2S0640
     1      GO TO 80                                                     CN2S0650
          IF (ISTG(NN1,ISEG) .EQ. ISTG(NN1,JSEG) .OR.                    CN2S0660
     1        ISTG(NN2,ISEG) .EQ. ISTG(NN2,JSEG)) GC TO 120              CN2S0670
 80       DIST=STG(LEN,ISEG)+STG(LEN,JSEG)                               CN2S0680
          IF (NS .EQ. 0) GC TO 100                                       CN2S0690
            DO 90 L=1,NS                                                 CN2S0700
            IF (JSECT .NE. NP1(L) .OR. KSECT .NE. NP2(L)) GO TO 90       CN2S0710
            IF (DIST .GT. SLEN(L)) GO TO 120                             CN2S0720
            NST=L          $      GC TO 110                              CN2S0730
 90         CONTINUE                                                     CN2S0740
100       NST=NS=NS+1                                                    CN2S0750
110       NP1(NST)=JSECT   $      NP2(NST)=KSECT                         CN2S0760
          NS1(NST)=ISEG    $      NS2(NST)=JSEG    $   SLEN(NST)=DIST    CN2S0770
120       CONTINUE                                                       CN2S0780
```

```
 130    CONTINUF
 140 CONTINUE

     NCON=0
     DO 170  I=1.NS
     NNFW=NP1(I)            $     NCLD=NP2(I)
     ISTG(MSECT,NS1(I))=ISTG(MSECT,NS2(I))=NNEW
     IF (NOLD .EQ. NNEW) GO TO 170
C       REPLACE NOLD BY NNEW IN THE SEGMENT TABLE
     IFOUND=0
        DO 150 J=1.NSEG
        K=ISTG(MSECT,J)     $     IF (K .NE. NOLD) GO TO 150
        ISTG(MSECT.J)=NNEW  $     IFOUND=1
 150    CONTINUF
     NCON=NCON+IFOUND
C       REPLACE NOLD BY NNEW IN THE REMAINDER OF THE REPLACEMENT TABLE
     IF (I .EQ. NS) GO TO 170
     IP1=I+1
        DO 160 J=IP1.NS
        IF (NP1(J) .NE. NOLD  .AND.  NP2(J) .NE. NOLD) GOTO 160
        IF (NP1(J) .EQ. NOLD) NP1(J)=NNEW
        IF (NP2(J) .EQ. NOLD) NP2(J)=NNEW
        IF (NP1(J) .LT. NP2(J)) GO TO 160
        K=NP1(J)            $     NP1(J)=NP2(J)     $     NP2(J)=K
 160    CONTINUE
 170 CONTINUE
     RETURN
     END
```

```
      SUBROUTINE FNDPTH(NTRIP,NORG,IP,NDSV,LNF,NPF,LNT,NPT,TIM,IPATH)

C        LATEST CHANGES --
C        JULY 21, 1976. MJI. ORIGINAL VERSION

C        THIS SUBROUTINE FINDS UP TO 5 SHORTEST PATHS CONNECTING PIECES
C        OF SECTION NTRIP TO NODE NORG OF PIECE IP.

      COMMON NSEG,ISTG(11,500)
      COMMON /NODDATA/ KNODES,NODNUM(300),XNOD(300),YNOD(300),
     1  NUMBER(300),NBRSEG(300)
      COMMON /TEMSTG/ DIST(300),TIME(300),LPREVN(300),NPREVS(300)
      DIMENSION LNF(5),LNT(5),NPF(5),NPT(5),TIM(5),IPATH(50,5)
      DATA MAXNDS/5/
      DATA NN1,NN2,MSECT/2,3,10/

      MAXSTG=LOCF(IPATH(1,2))-LOCF(IPATH(1,1))
      DO 10 I=1,KNODES
      IF (NORG .EQ. NODNUM(I)) LNORG=I
      DIST(I)=TIME(I)=0.
   10 LPREVN(I)=NPREVS(I)=0
      DO 40 I=1,MAXNDS
        DO 30 J=1,MAXSTG
   30   IPATH(J,I)=0
      NPT(I)=IP
   40 LNT(I)=LNORG

C        BUILD A TIME TREE FROM NORG
      CALL TREE(NORG,DIST,TIME,0.,LPREVN,NPREVS,1,KNODES,0)

C        FIND THE CLOSEST (IN TIME) POINTS IN EACH OF UP TO 5 (MAXNDS)
C        OTHER PIECES
      NDSV=0
      DO 90 I=1,NSEG
      JP=ISTG(MSECT,I)
      IF (MOD(JP,1000) .NE. NTRIP .OR. IP .EQ. JP) GO TO 90
      N1=ISTG(NN1,I)          $      N2=ISTG(NN2,I)
      L=IFIND(N1,NODNUM,KNODES)    $     L2=IFIND(N2,NODNUM,KNODES)
      IF (TIME(L) .GT. TIME(L2)) L=L2  $      T=TIME(L)
```

```
      JJ=1
      IF (NDSV .EQ. 0) GO TO 60
C        SEEK ANOTHER NODE IN THE SAME PIECE (JP).
        DO 50 J=1,NDSV
        JJ=J
        IF (NPF(J) .NE. JP) GO TO 50
        IF (T .GE. TIM(J)) 90,70
   50   CONTINUE

C        NO OTHER NODES FROM PIECE JP HAVE BEEN SAVED YET. SAVE THE
C        CURRENT NODE IF THERE IS ROOM OR IF IT IS REACHED IN A SHORTER
C        TIME THAN SOME NODE ALREADY SAVED.
      JJ=NDSV+1
      IF (NDSV .LT. MAXNDS) GO TO 60
      IF (T .GE. TMAX) GO TO 90
C        REPLACE THE POOREST TIME
      JJ=LTMAX                  $      GO TO 70

   60 NDSV=NDSV+1
   70 NPF(JJ)=JP    $     LNF(JJ)=L       $     TIM(JJ)=T
C        FIND THE WORST TIME
      TMAX=0.
        DO 80 J=1,NDSV
        IF (TIM(J) .LE. TMAX) GO TO 80
        TMAX=TIM(J)             $      LTMAX=J
   80   CONTINUE
   90 CONTINUE

C        SAVE THE PATH TO NORG FOR EACH PIECE
      DO 120 I=1,NDSV
      NP=0                      $      L=LNF(I)
  100 NP=NP+1
      J=IPATH(NP,I)=NPREVS(L)      $      L=LPREVN(L)
      IF (ISTG(MSECT,J) .EQ. NPT(I) .OR. L .EQ. LNORG) GO TO 120
      IF (NP .LT. MAXSTG) GO TO 100
      PRINT 110, MAXSTG,NPF(I),NPT(I)
  110 FORMAT(*0MORE THAN*,I5,* STEPS  NEEDED TO CONNECT PIECES*,I5,* AND
     1*,I5/* IF THIS IS CORRECT, INCREASE THE FIRST DIMENSION OF IPATH IF
```

```
     2N COMMON BLOCK TEMSTG. SUBROUTINES FNDPTH AND CONNST.*/* JOB TERMIFNDPT790
     3NATED*)                                                           FNDPT800
      PRINT 115, (NODNUM(LNF(K)),NPF(K),NODNUM(LNT(K)),NPT(K),TIM(K),   FNDPT810
     1  (IPATH(J,K),J=1,15),K=1,NDSV)                                   FNDPT820
  115 FORMAT(*0      FROM          TO*/*  NODE  PIECE  NODE  PIECE  TIME FNDPT830
     1  FIRST 15 SEGMENTS IN PATH*/(I6,I7,I6,I7,F8.3,15I5.0))           FNDPT840
      CALL EXIT                                                         FNDPT850
  120 CONTINUE                                                          FNDPT860
      RETURN                                                            FNDPT870
      END                                                               FNDPT880
```

```
      SUBROUTINE CONNST(NTRIP,NPIECE)                                   CNNST010
                                                                        CNNST020
C         LATEST CHANGES --                                             CNNST030
C         JULY 22, 1976. HJI.ORIGINAL VERSION.                          CNNST040
                                                                        CNNST050
C         THIS SUBROUTINE CONNECTS PIECES OF SECTION NTRIP WHICH COULD  CNNST060
C         NOT BE CONNECTED BY 1 OR 2 STEP PATHS.                        CNNST070
                                                                        CNNST080
      COMMON NSEG,ISTG(11,500)                                          CNNST090
      COMMON /NODDATA/ KNODES,NODNUM(300)                               CNNST100
      COMMON /TEMSTG/ XXX(300,4),NPF(25),NPT(25),LNF(25),LNT(25),       CNNST110
     1 TIM(25),IPATH(50,25)                                             CNNST120
      DATA NN1,NN2,MSECT/ 2,3,10/                                       CNNST130
                                                                        CNNST140
      ITER=0                                                            CNNST150
C         FIND ANY NODE IN THE SECTION                                  CNNST160
      DO 10 I=1,NSEG                                                    CNNST170
      IS=ISTG(MSECT,I)                                                  CNNST180
      IF (MOD(IS,1000) .NE. NTRIP) GO TO 10                             CNNST190
      NORG=ISTG(NN1,I)         $       GO TO 20                         CNNST200
   10 CONTINUE                                                          CNNST210
                                                                        CNNST220
   20 ITER=ITER+1                                                       CNNST230
C         FIND UP TO 5 NODES IN OTHER PIECES.                           CNNST240
      CALL FNDPTH(NTRIP,NORG,IS,NDSV,LNF(21),NPF(21),LNT(21),NPT(21),   CNNST250
     1  TIM(21),IPATH(1,21))                                            CNNST260
      LIM=NDSV        $       NDC=1                                     CNNST270
                                                                        CNNST280
C         USE EACH POINT SAVED AS THE STARTING POINT TO UP TO 5 OTHER   CNNST290
C         PIECES                                                        CNNST300
      DO 40 I=1,LIM                                                     CNNST310
      NNN=NODNUM(LNF(I+20))    $       IP=NPF(I+20)                     CNNST320
      CALL FNDPTH(NTRIP,NNN,IP,NDSV,LNF(NDC),NPF(NDC),LNT(NDC),NPT(NDC),CNNST330
     1  TIM(NDC),IPATH(1,NDC))                                          CNNST340
   40 NDC=NDC+NDSV                                                      CNNST350
      NDC=NDC-1                                                         CNNST360
                                                                        CNNST370
C         FIND THE CONNECTION WITH THE SMALLEST TRAVERSAL TIME.  ADD THE CNNST380
C         PATH SEGMENTS AND THE HIGHER NUMBERED PIECE TO THE LOWER      CNNST390
```

```
C          NUMBERED PIECE.                                              CNNST400
   50 TMIN=1.E20    $      LTM=0                                       CNNST410
      DO 60 I=1,NOC                                                     CNNST420
      IF (TIM(I) .GE. TMIN .OR. NPF(I) .EQ. NPT(I)) GO TO 60            CNNST430
      LTM=I         $      TMIN=TIM(I)                                  CNNST440
   60 CONTINUE                                                          CNNST450
      IF (LTM .GT. 0) GO TO 80                                          CNNST460
      PRINT 70                                                          CNNST470
   70 FORMAT(*0NO CONVERGENCE IN CCNNST*)                               CNNST480
      RETURN                                                            CNNST490
                                                                        CNNST500
   80 TIM(LTM)=2.E20                                                    CNNST510
      NPS=MIN0(NPT(LTM),NPF(LTM))                                       CNNST520
      NPL=MAX0(NPT(LTM),NPF(LTM))                                       CNNST530
      NPIECE=NPIECE-1                                                   CNNST540
      DO 90 I=1,NOC                                                     CNNST550
      IF (NPT(I) .EQ. NPL) NPT(I)=NPS                                   CNNST560
   90 IF (NPF(I) .EQ. NPL) NPF(I)=NPS                                   CNNST570
C          PUT SEGMENTS FROM HIGHER NUMBERED PIECE INTO LOWERED PIECE   CNNST580
      DO 100 I=1,NSEG                                                   CNNST590
  100 IF (ISTG(MSECT,I) .EQ. NPL) ISTG(MSECT,I)=NPS                     CNNST600
C          ADD CONNECTING PATH TO LOWER NUMBERED PIECE                  CNNST610
      DO 110 I=1,50                                                     CNNST620
      J=IPATH(I,LTM)          $       IF (J .EQ. 0) GO TO 120           CNNST630
  110 ISTG(MSECT,J)=NPS                                                 CNNST640
C          SEE IF ANY OTHER CONNECTIONS REMAIN.                         CNNST650
  120 DO 130 I=1,NOC                                                    CNNST660
      IF (NPF(I) .NE. NPS .OR. NPT(I) .NE. NPS) GO TO 50                CNNST670
  130 CONTINUE                                                          CNNST680
      IF (ITER .LT. 10 .AND. NPIECE .GT. 1) GO TO 20                    CNNST690
      IF (ITER .GE. 10) PRINT 145, ITER                                 CNNST700
  145 FORMAT(*0CONNST ITER*,I5)                                         CNNST710
      RETURN                                                            CNNST720
      END                                                               CNNST730
```

```
      SUBROUTINE CONNECT(NTRIP,NPIECE)                                  CNCT0010
                                                                        CNCT0020
C        LATEST CHANGES  --                                             CNCT0030
C        JUNE 16, 1976. HJI. ORIGINAL VERSION (INCOMPLETE)              CNCT0040
                                                                        CNCT0050
      COMMON NSEG,STG                                                   CNCT0060
      COMMON /TEMSTG/ NNFN(300),NNTN(300),NSFN(300),NSTN(300),TFN(300), CNCT0070
     1  TTN(300),DFN(300),DTN(300)                                      CNCT0080
      COMMON /NODDATA/ KNODES,NODNUM(300),XNOD(300),YNOD(300),          CNCT0090
     1  NUMNBR(300),NBRSEG(300)                                         CNCT0100
      DIMENSION ISTG(11,500), STG(11,500)                               CNCT0110
      DIMENSION ISECT(2),NNN(2)                                         CNCT0120
      EQUIVALENCE (STG,ISTG)                                            CNCT0130
      DATA NN1,NN2,LEN,NH,NSECT,MSECT/ 2,3,4,5,9,10/                    CNCT0140
                                                                        CNCT0150
      II=1                 $     NPIECE=1      $      ITRIP=NTRIP       CNCT0160
      DO 10 I=1,KNODES                                                  CNCT0170
   10 NNFN(I)=NNTN(I)=NSFN(I)=NSTN(I)=TFN(I)=TTN(I)=DFN(I)=DTN(I)=0.    CNCT0180
C        FIND THE NEXT SEGMENT IN SECTION ITRIP                         CNCT0190
   20 DO 30  I=II,NSEG                                                  CNCT0200
      II=I                 $    IF (ITRIP .EQ. ISTG(MSECT,I)) GO TO 40CNCT0210
   30 CONTINUE                                                          CNCT0220
      STOP 30                                                           CNCT0230
                                                                        CNCT0240
   40 NORG=ISTG(NN1,II)                                                 CNCT0250
      CALL TREF(NORG,DFN,TFN,1.,NNFN,NSFN,1,KNODES,ITRIP)               CNCT0260
      CALL TREE(NORG,DTN,TTN,1.,NNTN,NSTN,-1,KNODES,ITRIP)              CNCT0270
                                                                        CNCT0280
C        FLAG DISJOINT SEGMENTS                                         CNCT0290
      DJ=0.                                                             CNCT0300
      II=II+1              $     IF (II .GT. NSEG) GO TO 80             CNCT0310
      DO 70 I=II,NSEG                                                   CNCT0320
      IF (ITRIP .NE. ISTG(MSECT,I)) GO TO 70                            CNCT0330
C        SEGMENT I IS IN SECTION ITRIP.  IF IT IS CONNECTED TO NODE     CNCT0340
C        NORG, THEN EACH END POINT WILL HAVE A NON-ZERO NNFN OR NNTN.   CNCT0350
      J=ISTG(NN1,I)        $     IF (J .EQ. NORG) GO TO 70              CNCT0360
      LINE=IFIND(J,NODNUM,KNODES)       $       IF (LINE .LE. 0) STOP 70 CNCT0370
      IF (NNFN(LINE) .NE. 0  .OR. NNTN(LINE) .NE. 0) GO TO 70           CNCT0380
C        SEGMENT I IS DISJOINT FROM NORG. ADD 1000 TO ITS SECTION NUMBERCNCT0390
```

```
      ISTG(MSECT,I)=ISTG(MSECT,I)+1000             $     DJ=1.          CNCT0400
   70 CONTINUE                                                          CNCT0410
      IF (DJ .EQ. 0.) GO TO 80                                          CNCT0420
      NPIECE=NPIECE+1                                                   CNCT0430
C         SEE IF THE SEGMENTS DISJOINT FROM NORG ARE CONNECTED TO       CNCT0440
C         THEMSELVES                                                    CNCT0450
      ITRIP=ITRIP+1000                                                  CNCT0460
      GO TO 20                                                          CNCT0470
                                                                        CNCT0480
C         NO MORE DISJOINT PIECES ARE LEFT                              CNCT0490
   80 CONTINUE                                                          CNCT0500
      IF (NPIECE .EQ. 1) RETURN                                         CNCT0510
                                                                        CNCT0520
C         SCAN SEGMENT LIST TO FIND ALL SINGLE SEGMENTS WHICH CONNECT   CNCT0530
C         DISJOINT PIECES OF SECTION NTRIP                              CNCT0540
      DO 130 I=1,NSEG                                                   CNCT0550
      JSECT=MOD(ISTG(MSECT,I),1000)                                     CNCT0560
      IF (JSECT .EQ. NTRIP) GO TO 130                                   CNCT0570
      NNN(1)=ISTG(NN1,I)        $       NNN(2)=ISTG(NN2,I)              CNCT0580
         DO 110 J=1,2                                                   CNCT0590
         LINE=IFIND(NNN(J),NCONUM,KNODES)                               CNCT0600
         IF (LINE .LE. 0) STOP111                                       CNCT0610
         LIM=NUMNBR(LINE)       $       IF (LIM .LE. 1) GO TO 130       CNCT0620
            DO 100 K=1,LIM                                              CNCT0630
            ISEG=SHIFT(NBRSEG(LINE),10*(1-K)) .A. 1777B                 CNCT0640
            IF (ISEG .EQ. I) GO TO 100                                  CNCT0650
            ISECT(J)=ISTG(MSECT,ISEG)                                   CNCT0660
            IF (MOD(ISECT(J),1000) .EQ. NTRIP) GO TO 110                CNCT0670
  100       CONTINUE                                                    CNCT0680
C        END POINT DOES NOT LIE IN SECTION NTRIP                        CNCT0690
         GO TO 130                                                      CNCT0700
  110    CONTINUE                                                       CNCT0710
      IF (ISECT(1) .EQ. ISECT(2)) GO TO 130                             CNCT0720
C        SAVE CONCATENATED PIECE NUMBERS AS THE NEW SECTION NUMBER      CNCT0730
      IF (ISECT(1) .LT. ISECT(2)) GO TO 120                             CNCT0740
      ISTG(MSECT,I)=SHIFT(ISECT(2),20) .O. ISECT(1)                     CNCT0750
      GO TO 130                                                         CNCT0760
  120 ISTG(MSECT,I)=SHIFT(ISECT(1),20) .O. ISECT(2)                     CNCT0770
  130 CONTINUE                                                          CNCT0780
```

```
C         CONNECT PIECES, ASSIGNING LOWER PIECE NUMBER TO CONNECTED PIECE
      DO 160 I=1,NSEG
      J=ISTG(MSECT,I)
      IF (J .LT. 1000000) GO TO 160
      NOLD=J.A. 37777778      $      NNEW=SHIFT(J,-20)
      IF (NOLD .EQ. NNEW) GO TO 150
      ISTG(MSECT,I)=NNEW
         DO 140 J=1,NSEG
         K=ISTG(MSECT,J)      $     L=SHIFT(K,-20)   $   K=K .A. 37777778
         IF (K .NE. NOLD .AND. L .NE. NOLD) GO TO 140
         IF (K .EQ. NOLD) K=NNEW   $     IF (L .EQ. NOLD) L=NNEW
         IF (K .GT. L) ISTG(MSECT,J)=K .O. SHIFT(L,20)
         IF (K .LT. L) ISTG(MSECT,J)=L .O. SHIFT(K,20)
  140    CONTINUE
      GO TO 160
  150 ISTG(MSECT,I)=NNEW
  160 CONTINUE

C         COUNT REMAINING PIECES
      NCON=0                  $      NMAX=NTRIP+1000*(NPIECE-1)
      DO 170 J=NTRIP,NMAX,1000
         DO 165 I=1,NSEG
         IF (ISTG(MSECT,I) .NE. J) GO TO 165
         NCON=NCON+1          $      GO TO 170
  165    CONTINUE
  170 CONTINUE
      NPIECE=NCON             $      IF (NPIECE .LE. 1) RETURN

      CALL CON2ST(NTRIP,NCON)
      NPIECE=NPIECE-NCON      $      IF (NPIECE .LE. 1) RETURN
C         COUNT HOUSES ON EACH PIECE.  SAVE PIECE NUMBER IN NNFN AND
C         SAVE NUMBER OF HOUSES IN NNTN.
      DO 210 I=1,NPIECE
  210 NNFN(I)=NNTN(I)=0
      DO 240 I=1,NSEG
      K=ISTG(MSECT,I)         $      IF (MOD(K,1000) .NE. NTRIP) GO TO 240
         DO 230 J=1,NPIECE
         IF (NNFN(J) .GT. 0) GO TO 220
```

```
      NNFN(J)=K
      IF (ISTG(NSECT,I) .EQ. NTRIP) NNTN(J)=ISTG(NH,I)   $  GO TO 240
  220 IF (K .NE. NNFN(J)) GO TO 230
      IF (ISTG(NSECT,I) .EQ. NTRIP)  NNTN(J)=NNTN(J)+ISTG(NH,I)
      GO TO 240
  230 CONTINUE
   STOP 230
  240 CONTINUE
C     CHECK FOR NO HOUSES
   NMAX=0                    $      NCON=0
   DO 270  I=1,NPIECE
   IF (NNTN(I) .GT. 0) GO TO 260
   K=NNFN(I)                 $      NCON=NCON+1
C     CHANGE SECTION ASSIGNMENT TO ZERO.
      DO 250 J=1,NSEG
  250 IF (ISTG(MSECT,J) .EQ. K) ISTG(MSECT,J)=0
   GO TO 270
  260 IF (NNFN(I) .GT. NMAX) NMAX=NNFN(I)
  270 CONTINUE
   DO 280 I=1,NPIECE
   J=I                       $      IF (NNFN(I) .EQ. NTRIP) GO TO 290
  280 CONTINUE
   STOP 270
  290 IF (NNTN(J) .GT. 0) GO TO 310
C     PIECE NUMBERED NTRIP WAS DELETED, SO CHANGE THE HIGHEST
C     NUMBERED NON-EMPTY PIECE TO NTRIP.
   DO 300 I=1,NSEG
  300 IF (ISTG(MSECT,I) .EQ. NMAX) ISTG(MSECT,I)=NTRIP
  310 NPIECE=NPIECE-NCON
   IF (NPIECE .EQ. 1) RETURN
   CALL CONNST(NTRIP,NPIECE)
   RETURN
   END
```

```
      SUBROUTINE DISCON(NTRIP,KN,NODPTR,NODORD)                         DSCON010
                                                                        DSCON020
C         LATEST CHANGE  --                                             DSCON030
C         JUNE 18, 1976. HJI. ORIGINAL VERSION                          DSCON040
                                                                        DSCON050
      COMMON NSEG,ISTG(11,500)                                          DSCON060
      COMMON /NODDATA/ KNODES,NODNUM(300),XNOD(300),YNOD(300),          DSCON070
     1  NUMNBR(300),NBRSEG(300)                                         DSCON080
      DIMENSION NODPTR(1), NODORD(1)                                    DSCON090
      DATA NN1,NN2,NH,NSECT,MSECT/ 2,3,5,9,10/                          DSCON100
                                                                        DSCON110
      IDEL=0                                                            DSCON120
   10 IRP=0                                                             DSCON130
      DO 60 I=1,KN                                                      DSCON140
      IF (NODORD(I) .NE. 1) GO TO 60                                    DSCON150
C         FIND SEGMENT                                                  DSCON160
      LINE=NODPTR(I)                                                    DSCON170
      LIM=NUMNBR(LINE)        $     NBRS=NBRSEG(LINE)                   DSCON180
          DO 20 J=1,LIM                                                 DSCON190
          ISEG=SHIFT(NBRS,10*(1-J))  .A. 1777B                          DSCON200
          IF (ISTG(MSECT,ISEG) .EQ. NTRIP) GO TO 30                     DSCON210
   20     CONTINUE                                                      DSCON220
      PRINT 25, I,LINE,LIM,NBRS,(J,NODPTR(J),NODORD(J),J=1,KN)          DSCON230
   25 FORMAT(*0I=*,I3,*   LINE=*,I4,*   LIM=*,I3,*   NBRS=*,O20/        DSCON240
     1  *0       NODPTR  NODORD*/ (I4,2I8))                             DSCON250
      CALL DUMP(0,656008,4)                                             DSCON260
   30 IF (ISTG(NSECT,ISEG) .EQ. NTRIP  .AND.  ISTG(NH,ISEG)  .GT. 0)    DSCON270
     1  GO TO 60                                                        DSCON280
C         DELETE SEGMENT                                                DSCON290
      ISTG(MSECT,ISEG)=0      $     NODORD(I)=0                         DSCON300
      IDEL=IDEL+1                                                       DSCON310
      NODE=ISTG(NN1,ISEG)                                               DSCON320
      IF (NODE .EQ. NODNUM(LINE)) NODE=ISTG(NN2,ISEG)                   DSCON330
C         SEARCH FOR OTHER NODE LOCATION IN NODPTR ARRAY                DSCON340
          DO 40 J=1,KN                                                  DSCON350
          JJ=J            $     IF (NODNUM(NODPTR(J)) .EQ. NODE) GO TO 50 DSCON360
   40     CONTINUE                                                      DSCON370
      PRINT 45, I,NODE,         (J,NODPTR(J),NODORD(J),J=1,KN)          DSCON380
   45 FORMAT(*0I=*,I3,*   NODE=*,I4,*0      NODPTR  NODORD*/ (I4,2I8))  DSCON390
```

```
      CALL DUMP(0,656008,4)
   50 NODORD(JJ)=NODORD(JJ)-1
      IF (JJ .LT. I   .AND.   NODORD(JJ)  .EQ. 1) IRP=1
   60 CONTINUE
      IF (IRP .EQ. 1) GO TO 10
C
C         CLOSE UP NODPTR AND NODORD TABLES
      J=0
      DO 70  I=1,KN
      IF (NODORD(I)  .EQ. 0)  GO TO 70
      J=J+1                     $      IF (I .EQ. J) GO TO 70
      NODPTR(J)=NODPTR(I)       $      NODORD(J)=NODORD(I)
   70 CONTINUE
      KN=KN-IDEL
      RETURN
      END
```

```
      SUBROUTINE EPXP(KN,NCDPTR,NODORD,I)

C         LATEST CHANGES  --
C         JULY 30, 1976. HJI. REMOVED INITIAL PASS SELECTING CCD NODES.
C         JULY 27, 1976. HJI. EPXP TAKEN FROM CODING IN PHASE3.

C         EPXP FINDS GOOD ENTRY AND EXIT POINTS FOR THE CURRENT SECTION.

      COMMON /NODDATA/ KNODES,NODNUM(300)
      COMMON /TEMSTG/ DISTS(100),NNTF(300,3),DTF(300,3),TTF(300,3),
     1  NSTF(300,3),XXX,NTF(10,3)
      DIMENSION NODORD(100),NODPTR(100)
      KLIM=MINO(MAXO(KN/4,5),KN/2+1,10)
      KKEEP=6                  $      KLPK=MINO(KN,KLIM+KKEEP)
      BUFFER IN (7,1) (NNTF,XXX)
      IF (UNIT(7)) 236,236,236
  236 REWIND 7
          DO 240 J=1,10
  240     NTF(J,1)=NTF(J,2)=NTF(J,3)=0

C         FIND KLIM NODES WHICH ARE SUITABLE FOR ENTRY TO SECTION FROM
C         DUMP (J=1), ENTRY FROM GARAGE (J=2), AND EXIT FROM SECTION TO
C         DUMP (J=3).

          DO 290 J=1,3
             DO 250 K=1,KN
  250        DISTS(K)=TTF(NODPTR(K),J)

C         SORT THE TIMES (DISTS ARRAY) AND CARRY ALONG LINE NUMBERS
C         AND NODE ORDERS.
          CALL SHLSRT3(DISTS,NODPTR,NODORD,KN,1.)

C         KEEP THE CLOSEST NODES UNTIL KKEEP NODES HAVE BEEN SAVED,
C         PROVIDED THAT EACH NODE DOES NOT HAVE ANOTHER NODE IN THE
C         SECTION ON THE PATH TO THE DUMP OR GARAGE.
          JN=0
             DO 280 K=1,KLPK
             LINE=NODPTR(K)     $        NS=NODNUM(LINE)
             IF (JN .EQ. 0) GO TO 270
```

```
C        CHECK FOR OTHER NODES LYING ON THE PATH FROM THE SECTION TO
C        THE DUMP OR GARAGE.
               DO 260 L=1,KLIM
               IF (LINE .EQ. 0) GO TO 270
               LINE=NNTF(LINE,J)
                  DO 260 M=1,KN
                  IF (LINE .EQ. NODPTR(M)) GO TO 280
  260             CONTINUE

C        KEEP NODE
  270         JN=JN+1
              NTF(JN,J)=NS
  280         CONTINUE
  290      CONTINUE
  300 PRINT 310, I,((NTF(J,K),J=1,10),K=1,3)
  310 FORMAT(/*0GOOD ENTRY/EXIT NODES FOR SECTION*,I3/*0FROM DUMP   *,
     1  10I5.0/* FROM GARAGE*,10I5.0/*    TO DUMP   *,10I5.0)
      RETURN
      END
```

```
      SUBROUTINE CLOSE1(NTRIP,KN,KNX,NODPTR,NODORD)                     CLS10010
                                                                        CLS10020
C        LATEST CHANGES  --                                             CLS10030
C        MAR 10, 1977. HJI.  CHANGED TREATMENT OF U-TURNS               CLS10040
C        JULY 30, 1976. HJI. ADDED CODING TO DELETE FROM SECTION        CLS10050
C        SEGMENTS USED ON PREVIOUS CALL TO CLOSE OFF ORDER 1 NODES.     CLS10060
C        JUNE 18, 1976. HJI. ORIGINAL VERSION.                          CLS10070
                                                                        CLS10080
      COMMON NSEG,ISTG(11,500)                                          CLS10090
      COMMON /NODDATA/ KNODES,NODNUM(300),XNOD(300),YNOD(300),          CLS10100
     1  NUMNBR(300),NBRSEG(300)                                         CLS10110
      COMMON /TEMSTG/ IPATHN(32),IPATHS(32),LPREVN(300),NPREVS(300),    CLS10120
     1  DIST(300),TIME(300)                                             CLS10130
      DIMENSION NODORD(KN),NODPTR(KN), IPSSV(100)                       CLS10140
      DATA IPSSV,NSSV,NTRSV/102*0/                                      CLS10150
      DATA NSTR,NN1,NN2,LEN,NH,NSPD,NWAY,NRQF,MSECT/ 1,2,3,4,5,6,7,8,10/CLS10160
      DATA NSF/11/                                                      CLS10170
                                                                        CLS10180
      KNX=KN                                                            CLS10190
      IF (NTRIP .NE. NTRSV .OR. NSSV .EQ. 0) GO TO 8                    CLS10200
      DO 4 I=1,NSSV                                                     CLS10210
      ISTG(MSECT,IPSSV(I))=ISTG(NSF,IPSSV(I))=0                         CLS10220
    4 IPSSV(I)=0                                                        CLS10230
    8 NSSV=0                  $      NTRSV=NTRIP                        CLS10240
      DO 200 I=1,KN                                                     CLS10250
      IF (NODORD(I) .NE. 1) GO TO 200                                   CLS10260
C        FIND SEGMENT ATTACHED TO NODE                                  CLS10270
      LINE=NODPTR(I)          $      NBRS=NBRSEG(LINE)                  CLS10280
      LIM=NUMNBR(LINE)        $      IF (LIM .GT. 1) GO TO 40           CLS10290
C        STREET IS A DEAD END.  GENERATE A RETURN SEGMENT.              CLS10300
   10 ISTG(NSF,NBRS)=ISTG(NSF,NBRS)+1                                   CLS10310
      NODE=ISTG(NN2,NBRS)                                               CLS10320
      IF (NODE .EQ. NODNUM(LINE)) NODE=ISTG(NN1,NBRS)                   CLS10330
C        INCREMENT END POINT NODE ORDERS                                CLS10340
         DO 20 J=1,KN                                                   CLS10350
         JJ=J            $    IF (NODE .EQ. NODNUM(NODPTR(J))) GO TO 30 CLS10360
   20    CONTINUE                                                       CLS10370
      STOP 201                                                          CLS10380
   30 NODORD(JJ)=NODORD(JJ)+1      $      NODORD(I)=2                   CLS10390
```

```
      GO TO 200
C        FIND A PATH UP TO 4 STEPS LONG WHICH CLCSES OFF THE NODE.
   40 MAXLVL=4
C        FIND THE SEGMENT
         DO 50 J=1,LIM
         ISEG=SHIFT(NBRS,10*(1-J))  .A. 17778
         IF (ISTG(MSECT,ISEG) .EQ. NTRIP) GO TO 60
   50    CONTINUE
      STOP 501
   60 CONTINUE
C        SAVE SEGMENT ENDPOINTS AND ONE WAY INDICATOR
      NN1SV=ISTG(NN1,ISEG)     $     NN2SV=ISTG(NN2,ISEG)
      NWAYSV=ISTG(NWAY,ISEG)
      NORG=NODNUM(LINE)
      NOTHER=NN1SV             $      IF (NOTHER .EQ. NORG) NOTHER=NN2SV
      NPSV=0                   $      TMIN=1.E20
   70    DO 130 J=1,2
         IDIR=2*J-3       $     ISW1=J-1          $     ISW2=1-ISW1
         IF (NWAYSV .EQ. 1 .AND. ((NOTHER .EQ. NN1SV .AND. ICIR .EQ. -1)
     1      .OR. (NORG .EQ. NN1SV .AND. IDIR .EQ. 1))) GO TO 130
C        TEMPORARILY MODIFY ISEG
         ISTG(NN1,ISEG)=ISW2*NORG+ISW1*NOTHER
         ISTG(NN2,ISEG)=ISW1*NORG+ISW2*NOTHER
         ISTG(NWAY,ISEG)=1
C        MINIMIZE TIME OF TRAVEL BACK TO SECTION
            DO 75 K=1,KNODES
            DIST(K)=TIME(K)=0.
   75       LPREVN(K)=NPREVS(K)=0
         CALL TREE(NORG,DIST,TIME,0.,LPREVN,NPREVS,IDIR,MAXLVL,0)
C        SEE IF ANY NODE REACHED IN MAXLVL OR FEWER STEPS LIES IN THE
C        SECTION
           DO 120 K=1,KNX
           L=NODPTR(K)
           M=NPREVS(L)         $      IF (M .EQ. 0) GO TO 120
C        FOLLOW PATH BACK TO ENTRY TO SECTION
   80      IF (ISTG(MSECT,M) .NE. NTRIP) GO TO 100
C        SEGMENT IS IN SECTION.  BACK UP ONE MORE SEGMENT.
   90      L=LPREVN(L)         $      M=NPREVS(L)
```

```
          IF (K .GT. 0) GO TO 80                                        CLS10790
C           DEBUG PRINT                                                 CLS10800
      PRINT 94                                                          CLS10810
   94 FORMAT(*0 LINE   NODE   NXND   NXSG    DIST    TIME   SECT*/)      CLS10820
      DO 96 L=1,KNODES                                                  CLS10830
   96 IF (NPREVS(L) .NE. 0) PRINT 98, L,NODNUM(L),LPREVN(L),NPREVS(L),   CLS10840
     1  DIST(L),TIME(L),ISTG(MSECT,NPREVS(L))                           CLS10850
   98 FORMAT(4I6,2F7.2,I6)                                              CLS10860
      CALL DUMP(0,70000B,4)                                             CLS10870
  100       IF (TMIN .LE. TIME(L)) GO TO 120                            CLS10880
            TMIN=TIME(L)      $     NPSV=0          $     LF=L          CLS10890
               DO 110 II=1,MAXLVL                                       CLS10900
               IPATHS(II)=M  $      L=LPREVN(L)                         CLS10910
               IPATHN(II)=L  $      M=NPREVS(L)                         CLS10920
               IF (ISTG(MSECT,M) .EQ. NTRIP) GO TO 90                   CLS10930
               NPSV=NPSV+1   $      IF (L .EQ. LINE) GO TO 120          CLS10940
  110          CONTINUE                                                 CLS10950
  120       CONTINUE                                                    CLS10960
  130    CONTINUE                                                       CLS10970
      IF (NPSV .GT. 0 .OR. NWAYSV .NE. 1) GO TO 140                     CLS10980
C        A ONE WAY STREET COULD NOT BE CLOSED OFF IN MAXLVL STEPS.      CLS10990
C        DOUBLE MAXLVL AND TRY AGAIN.                                   CLS11000
      MAXLVL=2*MAXLVL          $     IF (MAXLVL .LE. 32) GO TO 70       CLS11010
      STOP 1401                                                         CLS11020
C        RESTORE SEGMENT ISEG                                           CLS11030
  140 ISTG(NN1,ISEG)=NN1SV     $     ISTG(NN2,ISEG)=NN2SV               CLS11040
      ISTG(NWAY,ISEG)=NWAYSV                                            CLS11050
      IF (NPSV .GT. 0) GO TO 150                                        CLS11060
      NPSV=1                   $     IPATHS(1)=ISEG                     CLS11070
C        CLOSE OFF NODE BY ADDING A U-TURN                              CLS11080
      NBRS=ISEG                $     GO TO 10                           CLS11090
                                                                        CLS11100
  150 NODORD(I)=2                                                       CLS11110
         DO 160 K=1,KNX                                                 CLS11120
         IF (NODPTR(K) .NE. LF) GO TO 160                               CLS11130
         NODORD(K)=NODORD(K)+1      $     GO TO 170                     CLS11140
  160    CONTINUE                                                       CLS11150
      STOP 1601                                                         CLS11160
  170    DO 180 K=1,NPSV                                                CLS11170
```

```
      NSSV=NSSV+1           $       IPSSV(NSSV)=IPATHS(K)                      CLS11180
      ISTG(NSF,IPATHS(K))=1                                                    CLS11190
      ISTG(MSECT,IPATHS(K))=NTRIP    $      IF (K .EQ. NPSV) GO TO 180        CLS11200
      KNX=KNX+1                                                                CLS11210
      NODPTR(KNX)=IPATHN(K)      $     NODORD(KNX)=2                           CLS11220
180   CONTINUE                                                                 CLS11230
200 CONTINUE                                                                   CLS11240
    RETURN                                                                     CLS11250
    END                                                                        CLS11260
```

```
      SUBROUTINE GENDM(ONDMTX,NODD,NODPTR,NTRIP)                        GENDM010
                                                                        GENDM020
C        LATEST CHANGES  --                                             GENDM030
C        JULY 28, 1976. HJI. ORIGINAL VERSION.                          GENDM040
                                                                        GENDM050
      COMMON /NODDATA/ KNODES,NODNUM(300)                               GENDM060
      COMMON /TEMSTG/ NN(300),NS(300),TIME(300),DIST(300)               GENDM070
      DIMENSION ONDMTX(NODD,NODD), NODPTR(100)                          GENDM080
                                                                        GENDM090
      DO 30 I=1,NODD                                                    GENDM100
      NORG=NODNUM(NODPTR(I))                                            GENDM110
         DO 10 J=1,KNODES                                               GENDM120
         DIST(J)=TIME(J)=0.                                             GENDM130
   10    NN(J)=NS(J)=0                                                  GENDM140
      CALL TREE(NORG,DIST,TIME,1.,NN,NS,1,KNODES,NTRIP)                 GENDM150
         DO 20 J=1,NODD                                                 GENDM160
   20    ONDMTX(I,J)=DIST(NODPTR(J))                                    GENDM170
   30 CONTINUE                                                          GENDM180
      RETURN                                                            GENDM190
      END                                                               GENDM200
```

```
      SUBROUTINE SELCRD(NNN,NODD,DIST,IPART,IDIR)

C        LATEST CHANGES  --
C        JULY 28, 1976. HJI. DOUBLE SUBSCRIPTED DISTANCE MATRIX USED.
C        APR 21, 1973.  HJI. ADAPTED FOR CDC6600 FROM IBM/360 VERSION.

      REAL MIND, IEXTD
      DIMENSION DIST(NNN,NNN), IPART(32), MIND(32), MINJ(32)

      DO 20 I=1,NODD
      MIND(I)=1000000
      MINJ(I)=0
   20 IPART(I)=0

C   FIND NEAREST NEIGHBOR FOR EACH ROW I OF THE DISTANCE MATRIX
      DO 60 I=1,NODD
      DO 40 J=1,NODD
      IF (I .EQ. J) GO TO 40
      D=DIST(I,J)                $      IF (D .GE. MIND(I)) GO TO 40
      MINJ(I)=J                  $     MIND(I)=D
   40 CONTINUE
   60 CONTINUE

C   FIND N-TH LARGEST (IF IDIR =-1) OR SMALLEST (IF IDIR=+1) NEAREST
C   NEIGHBOR DISTANCE
      DO 200 N=1,NODD
      IEXTI=0
      IEXTD=(1+IDIR)*1000000
      DO 80 I=1,NODD
      IF (IPART(I) .GT. 0  .OR.  IDIR*(MIND(I)-IEXTD) .GE. 0) GO TO 80
      IEXTD=MIND(I)
      IEXTI=I
   80 CONTINUE
      IF (IEXTD .EQ. (1+IDIR)*1000000  .AND.  IEXTI .EQ. 0) GO TO 200
  120 ID=MINJ(IEXTI)
      IPART(IEXTI)=ID
      IPART(ID)=IEXTI

C   IF PART OF THE MINIMUM DISTANCE PAIR JUST FOUND IS PART OF AN
```

```
C     UNUSED PAIR. FIND NEW PAIRS WHICH DO NOT USE NODES ALREACY IN USE.  SELOR400
      DO 160  I=1,NODD                                                    SELOR410
      IF (IPART(I) .GT. 0) GO TO 160                                      SELOR420
      IF (IPART(MINJ(I)) .EQ. 0) GO TO 160                                SELOR430
      MIND(I)=1000000                                                     SELOR440
      DO 140  J=1,NODC                                                    SELOR450
      IF (IPART(J) .GT. 0  .OR.  I .EQ. J) GO TO 140                      SELOR460
      D=DIST(I,J)            $     IF (D .GE. MIND(I)) GO TO 140          SELOR470
      MIND(I)=D              $     MINJ(I)=J                              SELOR480
  140 CONTINUE                                                            SELOR490
  160 CONTINUE                                                            SELOR500
  200 CONTINUE                                                            SELOR510
      RETURN                                                              SELOR520
      END                                                                 SELOR530
```

```
      SUBROUTINE PATH(ISTPO,ISTPN,II,KK,JJ,NG)
C
C        LATEST CHANGES  --
C        JULY 31, 1976. HJI. DISTANCE MATRICES TYPED REAL TC INTERFACE
C        PATH WITH PHASE3.
C        MAY 7, 1973.
C
      COMMON /PART/ NEDGE,IPART(32)
      COMMON /STP/NNPW,NE,NES,ISTP(32),JSTP(32)
      DIMENSION ISTPO(2,32,32), ISTPN(2,32,32)
C
      IDIM=2                      $      JDIM=LOCF(JSTP)-LOCF(ISTP)
      DO 10 L=1,JDIM
   10 ISTP(L)=0
      NE=0
      I=II
      J=KK
      DO 100 L=1,2
      MID=NE+1
      ISTP(NE+1)=I
      ISTP(NE+2)=IPART(I)
      NE=NE+2
      NW=1
   20 IWD=ISTPO(NW,I,J)
   40 IF (IWD .GT. 0) GO TO 60
      NW=NW+1
      IF (NW .LE. IDIM) GO TO 20
      GO TO 80
   60 NIWD=IWD/(NEDGE+1)
      M=IWD-NIWD*(NEDGE+1)
      ISTP(NE+1)=M
      ISTP(NE+2)=IPART(M)
      NE=NE+2
      IWD=NIWD
      GO TO 40
   80 I=J
      J=JJ
  100 CONTINUE
      NE=NE+1
```

```
      ISTP(NE)=JJ                                                       PATH0400
C                                                                       PATH0410
C     ISTP ARRAY CONTAINS NE ENTRIES WHICH ARE THE PATH FROM II TO JJ   PATH0420
C                                                                       PATH0430
      MIDM=MID-1                                                        PATH0440
      MIDP=MID+1                                                        PATH0450
C                                                                       PATH0460
C     CHECK FIRST HALF OF PATH AGAINST SECOND HALF FOR REPEATS          PATH0470
C                                                                       PATH0480
      NG=1                                                              PATH0490
      DO 120 I=1,MIDM                                                   PATH0500
      DO 120 J=MIDP,NE                                                  PATH0510
      IF (I .EQ. 1   .AND.  J .EQ. NE)GO TO 120                         PATH0520
      IF (ISTP(I) .EQ. ISTP(J)) RETURN                                  PATH0530
  120 CONTINUE                                                          PATH0540
C                                                                       PATH0550
C     REBUILD PACKED PATH                                               PATH0560
C                                                                       PATH0570
      NG=0                                                              PATH0580
      DO 160 L=1,IDIM                                                   PATH0590
      KI=2*(L-1)*NNPW+3                                                 PATH0600
      KF=MIN0(NE,KI+2*NNPW)-2                                           PATH0610
      ISTPN(L,II,JJ)=0                                                  PATH0620
      IWD=0                                                             PATH0630
      IF (KF .LT. KI)GO TO 160                                          PATH0640
        DO 140 K=KI,KF,2                                                PATH0650
        M=KF+KI-K                                                       PATH0660
  140   IWD=IWD*(NEDGE+1)+ISTP(M)                                       PATH0670
  160 ISTPN(L,II,JJ)=IWD                                                PATH0680
      IF ((NE-3)/2 .GT. IDIM*NNPW) NG=-1                                PATH0690
      RETURN                                                            PATH0700
      END                                                               PATH0710
```

```
      SUBROUTINE NEXTM(MTXO,MTXN,NOCC,ISTPO,ISTPN,NRES)                 NEXTM010
C                                                                       NEXTM020
C        LATEST CHANGES  --                                             NEXTM030
C        JULY 31, 1976. HJI. DISTANCE MATRICES TYPEC REAL TC INTERFACE  NEXTM040
C        NEXTM WITH PHASE3.                                             NEXTM050
C        APR 15, 1974. HJI.  ADAPTED FOR CDC6600 FROM IBM/360 VERSION.  NEXTM060
C                                                                       NEXTM070
      REAL MAXD,MAXP,MTXO,MTXN,MPR                                      NEXTM080
      COMMON /PART/ NEDGE,IPART(32)                                     NEXTM090
      COMMON /STP/NNPW,NE,NES,ISTP(32),JSTP(32)                         NEXTM100
      DIMENSION ISTPO(2,32,32), ISTPN(2,32,32), MTXN(32,32), MTXO(32,32)NEXTM110
      NRES=0                                                            NEXTM120
      NEDGE=NOCD                                                        NEXTM130
C                                                                       NEXTM140
C    DO CYCLES FIRST (DIAGONAL OF NEW MATRIX)                           NEXTM150
C                                                                       NEXTM160
      MAXD=0                                                            NEXTM170
      DO 160 I=1,NEDGE                                                  NEXTM180
      MAXP=MTXO(I,I)                                                    NEXTM190
      MTXN(I,I)=MAXP                                                    NEXTM200
      ISTPN(1,I,I)=ISTPO(1,I,I)    $      ISTPN(2,I,I)=ISTPO(2,I,I)     NEXTM210
      DO 140  J=1,NEDGE                                                 NEXTM220
      IF (I .EQ. J) GO TO 140                                           NEXTM230
      MPR=MTXO(I,J)+MTXO(J,I)                                           NEXTM240
      IF (MPR .LE. MAXP)GO TO 140                                       NEXTM250
C                                                                       NEXTM260
C    NEW CYCLE IS MORE PROFITABLE THAN OLD CYCLE                        NEXTM270
C                                                                       NEXTM280
      CALL PATH(ISTPO,ISTPN,I,J,I,NG)                                   NEXTM290
      IF (NG .NE. 0) GO TO 140                                          NEXTM300
      MTXN(I,I)=MPR                                                     NEXTM310
      MAXP=MPR                                                          NEXTM320
      IF (MPR .LE. MAXD) GO TO 140                                      NEXTM330
      MAXD=MPR                                                          NEXTM340
      NES=NE                                                            NEXTM350
         DO 120 K=1,NES                                                 NEXTM360
  120    JSTP(K)=ISTP(K)                                                NEXTM370
  140 CONTINUE                                                          NEXTM380
  160 CONTINUE                                                          NEXTM390
```

```
C                                                                       NEXTM400
C     NEW DIAGONAL HAS BEEN GENERATED                                   NEXTM410
C                                                                       NEXTM420
      IF (MAXD .LE. 1E-10) GO TO 180                                    NEXTM430
C                                                                       NEXTM440
C     A PROFITABLE CYCLE HAS BEEN FOUND                                 NEXTM450
C                                                                       NEXTM460
      NRES=1                                                            NEXTM470
      RETURN                                                            NEXTM480
  180 CONTINUE                                                          NEXTM490
      DO 240 I=1,NEDGE                                                  NEXTM500
      DO 240 J=1,NEDGE                                                  NEXTM510
      IF (I .EQ. J) GO TO 240                                           NEXTM520
      MAXP=MTXC(I,J)                                                    NEXTM530
      ISTPN(1,I,J)=ISTPO(1,I,J)    $      ISTPN(2,I,J)=ISTPO(2,I,J)     NEXTM540
      MTXN(I,J)=MAXP                                                    NEXTM550
         DO 220 K=1,NEDGE                                               NEXTM560
         IF (I .EQ. K  .OR. K .EQ. J) GO TO 220                         NEXTM570
         MPR=MTXO(I,K)+MTXO(K,J)                                        NEXTM580
         IF (MPR .LE. MAXP)GO TO 220                                    NEXTM590
         CALL PATH(ISTPO,ISTPN,I,K,J,NG)                                NEXTM600
         IF (NG .NE. 0) GO TO 220                                       NEXTM610
         MTXN(I,J)=MPR                                                  NEXTM620
         MAXP=MPR                                                       NEXTM630
  220    CONTINUE                                                       NEXTM640
  240 CONTINUE                                                          NEXTM650
C                                                                       NEXTM660
C     NEW MATRIX HAS BEEN GENERATED                                     NEXTM670
C                                                                       NEXTM680
      RETURN                                                            NEXTM690
      END                                                               NEXTM700
```

```
      SUBROUTINE SOLV(DIST,NODD,IERR)
C
C        LATEST CHANGES  --
C        JULY 31. 1976. HJI. DISTANCE MATRICES TYPED REAL TO INTERFACE
C        SOLV WITH PHASE3.
C        APR 15. 1974. HJI.  ADAPTED FOR CDC6600 FROM IBM/360 VERSION.
C
      REAL MTX1,MTX2
      COMMON /PART/ NEDGE, IPART(32)
      COMMON /STP/NNPW,NE,NES,ISTP(32),JSTP(32)
      COMMON /TEMSTG/ MTX1(32,32),MTX2(32,32),ISTP1(2,32,32),
     1  ISTP2(2,32,32)
      DIMENSION DIST(NODD,NODD)
C
      IERR=0
      NCH=0                 $     NNPW=14.449/ALOG10(NODD+1.)
      NN=2
      DO 40 K=1,6
      LIMK=K-1
      IF (NN .GT. NODD) GO TO 60
   40 NN=2*NN
   60 CONTINUE
C
C    FORM TWO-STEP DISTANCE MATRIX
C
  120 DO 220 I=1,NODD
      DO 220 J=1,NODD
      MTX1(I,J)=0.          $     MTX2(I,J)=0.
      DO 210 K=1,3
      ISTP1(K,I,J)=0
  210 ISTP2(K,I,J)=0
      IF (I .EQ. J) GO TO 220
      MTX1(I,J)=-1000.
      IF (J .EQ. IPART(I)) GO TO 220
      MTX1(I,J)=DIST(I,IPART(I))-DIST(IPART(I),J)
  220 CONTINUE
C
      DO 300 I=1,LIMK
      IF (MOD(I,2) .EQ. 0) GO TO 240
```

```
      CALL NEXTM(MTX1,MTX2,NODD,ISTP1,ISTP2,NRES)
      IF (NRES .NE. 0) GO TO 400
      GO TO 300
  240 CALL NEXTM(MTX2,MTX1,NODD,ISTP2,ISTP1,NRES)
      IF (NRES .NE. 0) GO TO 400
  300 CONTINUE
      RETURN
C
  400 CONTINUE
      NCH=NCH+1
      IF (NCH .LT. 20) GO TO 440
      PRINT 420, NCH
  420 FORMAT(*0NCH = *, I3, *   QUITTING*)
      IERR=1         $        RETURN
  440 DO 460 I=2,NES,2
      IPART(JSTP(I))=JSTP(I+1)
      IPART(JSTP(I+1))=JSTP(I)
  460 CONTINUE
      GO TO 120
      END
```

```
      SUBROUTINE TRAVEL(IST.ISP.NSEG.NN1.NN2.NSG,NWAY,NTIMES.AN,IPN,IPS,TRVL0010
     1  IERR)                                                            TRVL0020
                                                                         TRVL0030
C        LATEST CHANGES  --                                              TRVL0040
C        MAR 10.1977. HJI.   REMOVED SECOND PASS U-TURN PROCESSING.      TRVL0050
C        ALSO, INCREASED DIMENSIONS TO 200.                              TRVL0060
C        DEC 20. 1976. HJI.  ADDED NSG,IPS,IERR TO CALLING SEQUENCE.     TRVL0070
                                                                         TRVL0080
      DIMENSION KLINE(200),KTIMES(200),NEXTII(200),KS(200)               TRVL0090
      DIMENSION NN1(1),NN2(1),NSG(1),NWAY(1),NTIMES(1),IPN(1),IPS(1)     TRVL0100
      DATA KS/200*0/, MAXKS/100/, KSDIM/200/                             TRVL0110
                                                                         TRVL0120
      IERR=IPT1=1              $     M=KSDIM        $     IPT2=NSEG      TRVL0130
   10 IF(IPT2.LT.IPT1) GO TO 40                                          TRVL0140
      IF(NTIMES(IPT1).EQ.1) GO TO 15                                     TRVL0150
      IPT1=IPT1+1                                                        TRVL0160
      GO TO 10                                                           TRVL0170
   15 IF(NTIMES(IPT2).EQ.2) GO TO 30                                     TRVL0180
      IPT2=IPT2-1                                                        TRVL0190
      GO TO 10                                                           TRVL0200
   30 ISAVE1=NN1(IPT1)                                                   TRVL0210
      ISAVE2=NN2(IPT1)                                                   TRVL0220
      ISAVE3=NSG(IPT1)                                                   TRVL0230
      ISAVE4=NWAY(IPT1)                                                  TRVL0240
      ISAVE5=NTIMES(IPT1)                                                TRVL0250
      NN1(IPT1)=NN1(IPT2)      $     NN1(IPT2)=ISAVE1                    TRVL0260
      NN2(IPT1)=NN2(IPT2)      $     NN2(IPT2)=ISAVE2                    TRVL0270
      NSG(IPT1)=NSG(IPT2)      $     NSG(IPT2)=ISAVE3                    TRVL0280
      NWAY(IPT1)=NWAY(IPT2)    $     NWAY(IPT2)=ISAVE4                   TRVL0290
      NTIMES(IPT1)=NTIMES(IPT2)    $     NTIMES(IPT2)=ISAVE5             TRVL0300
      IPT1=IPT1+1              $     IPT2=IPT2-1                         TRVL0310
      GO TO 10                                                           TRVL0320
                                                                         TRVL0330
   40 DO 55 K=1.NSEG                                                     TRVL0340
   55 KTIMES(K)=NTIMES(K)                                                TRVL0350
                                                                         TRVL0360
      DO 205 KK=1,KSDIM                                                  TRVL0370
  205 KS(KK)=0                                                           TRVL0380
                                                                         TRVL0390
```

```
      IPN(1)=IST              $      NN=1      $      KLINE(1)=NEXTII(1)=0
   65 II=1

   90 DO 60 I=II,NSEG
      IF (KTIMES(I) .LT. 1  .OR.  KLINE(NN) .EQ. I) GO TO 60
      IF(NN1(I).EQ.IPN(NN)) GO TO 70
      IF (NWAY(I) .EC. 1  .OR.  NN2(I) .NE. IPN(NN)) GO TO 60
      IPS(NN)=NSG(I)
      NN=NN+1
      IPN(NN)=NN1(I)          $      KLINE(NN)=I
      NEXTII(NN)=I+1
      KTIMES(I)=KTIMES(I)-1
      GO TO 65

   70 IPS(NN)=NSG(I)
      NN=NN+1
      IPN(NN)=NN2(I)          $      KLINE(NN)=I
      NEXTII(NN)=I+1
      KTIMES(I)=KTIMES(I)-1
      GO TO 65
   60 CONTINUE

      I=KLINE(NN)
      IF (KTIMES(I) .EQ. 0) GO TO 75
      IPS(NN)=IPS(NN-1)
      NN=NN+1                 $      IPN(NN)=IPN(NN-2)
      KLINE(NN)=I             $      NEXTII(NN)=NSEG+1
      KTIMES(I)=KTIMES(I)-1
      GO TO 65

   75 IF (IPN(NN) .NE. ISP) GO TO 85
      DO 80 L=1,NSEG
      IF(KTIMES(L).EQ.0) GO TO 80
      IPN(M)=L                $      M=M-1
   80 CONTINUE
      IF (M .LT. KSDIM) GO TO 185
      IERR=0
  201 CONTINUE
      IF (IERR .GT. 0) PRINT 81, KS
```

```
 81 FORMAT(//* KS ARRAY*/ (20I6))
    IF (IERR .GT. 0) PRINT 900, IST,ISP,NSEG,(I,NN1(I),NN2(I),NSG(I),
   1  NWAY(I),NTIMES(I),I=1,NSEG)
900 FORMAT(*1TRAVEL ARGUMENTS*/*0ISTART=*,I5,*      ISTOP=*,I5,
   1 *      NSEG=*,I5/ 1H0,8X,*NN1    NN2    NSG   NWAY TIMES*// (6I6))
    IF (IERR .GT. () PRINT 910, NN,(I,IPN(I),IPS(I),I=1,100)
910 FORMAT(*1NN=*,I5/1H0,8X,*IPN    IPS*// (3I6))
    RETURN

 85 I=KLINE(NN)
    IF (I .LE. 0) GO TO 201
    KTIMES(I)=KTIMES(I)+1
    II=NEXTII(NN)              $      KS(NN)=KS(NN)+1
    IF (KS(NN) .GT. MAXKS) GO TO 201
180 NN=NN-1
    IF(II.LE.NSEG) 90,85
185 M=M+1
186 I=KLINE(NN)
    IF (I .LE. 0) GO TO 201
    KTIMES(I)=KTIMES(I)+1
    II=NEXTII(NN)              $      KS(NN)=KS(NN)+1
    IF (KS(NN) .GT. MAXKS) GO TO 201
210 NN=NN-1
    DO 83 I=M,KSDIM
    L=IPN(I)
    IF (NN1(L) .EQ. IPN(NN)  .OR.  NN2(L) .EQ. IPN(NN)) GO TO 84
 83 CONTINUE
    GO TO 186
 84 M=KSDIM
    IF(II.LE.NSEG) 90,85
    END
```

```
      SUBROUTINE OPTPATH(NNP,NSP,TRTY,N,ISC)

C        LATEST CHANGES  --
C        MAR 10, 1977. HJI.  ADDED CODING TO REMOVE STREETS TRAVELLED
C        TWICE IN THE SAME DIRECTION.
C        FEB 18, 1977. HJI.  ADDED U-TURN LIMITING.
C        JAN 31, 1977. HJI.  ORIGINAL VERSION.

      COMMON NSEG,ISTG(11,500)
      COMMON /TEMSTG/ DISTS(100),LNXN(450,2),DIST(450,2),TIME(450,2),
     1  NNXS(450,2)
      DIMENSION NNP(200),NSP(200),TRTY(200),STG(11,500)
      EQUIVALENCE (STG,ISTG)
         DATA NSTR,LEN,NH,NWAY,NSC,NSF/1,4,5,7,9,11/

      NM1=N-1

C        SET TRAVEL TYPE
      DO 20 I=1,NM1
      TRTY(I)=1HT             $     J=NSP(I)
      IF (ISTG(NSC,J) .NE. ISC  .OR.  ISTG(NSF,J) .EQ. 0  .OR.
     1 ISTG(NH,J) .EQ. 0) GO TO 20
      TRTY(I)=1HC             $     IF (ISTG(NSF,J) .GE. 2) ISTG(NSF,J)=0
   20 CONTINUE

C        FIND FIRST COLLECTION SEGMENT
      DO 30 I=1,NM1
      II=I                    $     IF (TRTY(I) .EQ. 1HC) GO TO 40
   30 CONTINUE
C        MOVE FIRST COLLECTION SEGMENT TO FRONT OF TRIP LIST.
   40 J=1
      DO 50 I=II,NM1
      NNP(J)=NNP(I)           $     NSP(J)=NSP(I) $    TRTY(J)=TRTY(I)
      IF (TRTY(J) .EQ. 1HC)  IF=J
   50 J=J+1
      NNP(J)=NNP(N)           $     N=MINO(J,IF+3)      $     NM1=N-1

C        OPTIMIZE TRAVEL STRETCHES
C        FIND START SEGMENT OF NEXT TRAVEL STRETCH
```

```
      IF=1
   60 DO 70 I=IF,NM1
      II=I                    $     IF (TRTY(I) .EQ. 1HT) GO TO 80
   70 CONTINUE
      GO TO 200

C        FIND END SEGMENT OF TRAVEL STRETCH
   80 DO 90 I=II,NM1
      IF=I                    $     IF (TRTY(I) .EQ. 1HC) GO TO 100
   90 CONTINUE
      GO TO 200

  100 IS=IF-II                $     IF (IS .EQ. 1) GO TO 60
      NS=0                    $     IF (NNP(II) .EQ. NNP(IF)) GO TO 115
C        FIND THE SHORTEST (TIME) TRAVEL FROM NNF(II) TO NNF(IF)
      DO 110 I=1,900
      DIST(I.1)=TIME(I.1)=0.
  110 LNXN(I.1)=NNXS(I.1)=0
      CALL TREE(NNP(IF),DIST,TIME,0.,LNXN,NNXS,-1,IS+3,0)
      CALL TRACE(NNP(II),NNP(IF),LNXN,NNXS,NNXS(1,2),LNXN(1,2),NS)
C        TEST FOR U-TURNS
      IF (NSP(II-1) .NE. NNXS(1.2)  .AND.  NSP(IF) .NE. NNXS(NS,2))
     1  GO TO 115
C        SEE IF U-TURNS CAUSE APPRECIABLE SAVINGS IN DISTANCE
      DO=CUMDIS(NSP(II),IS)   $    DN=CUMDIS(NNXS(1,2),NS)
      IF (DO-DN .LT. 0.5  .AND.  DO .LT. 3.*DN) GO TO 60
  115 IDEL=NS-IS              $    NEWN=N+IDEL
      IF (IDEL) 140.160.120
C        MAKE ROOM FOR ADDITIONAL SEGMENTS
  120 NNP(NEWN)=NNP(N)        $    LIM=N-IF
      DO 130 I=1.LIM
      NNP(NEWN-I)=NNP(N-I)    $    NSP(NEWN-I)=NSP(N-I)
  130 TRTY(NEWN-I)=TRTY(N-I)
      GO TO 160
C        CLOSE UP PATH LIST
  140 DO 150 I=IF,N
      NNP(I+IDEL)=NNP(I)      $    NSP(I+IDEL)=NSP(I)
  150 TRTY(I+IDEL)=TRTY(I)
C        MOVE NEW PATH INTO LIST
```

```
  160 IF (NS .EQ. 0) GO TO 180                                          OPTP0790
      DO 170 I=1,NS                                                     OPTP0800
      NNP(I+II-1)=LNXN(I,2)       $      NSP(I+II-1)=NNXS(I,2)          OPTP0810
  170 TRTY(I+II-1)=1HT                                                  OPTP0820
  180 IF=IF+IDEL              $      N=NEWN          $      NM1=N-1     OPTP0830
                                                                        OPTP0840
C        LOOK FOR ANCTHER TRAVEL STRETCH.                               OPTP0850
      GO TO 60                                                          OPTP0860
                                                                        OPTP0870
C        FIND THE LONGEST DOUBLY TRAVELLED PIECE.                       OPTP0880
  200 DO=0.                   $      NM1=N-1         $      NM3=N-3     OPTP0890
      DO 260 I=1,NM3                                                    OPTP0900
      IF (TRTY(I) .EC. 1HC) GO TO 260                                   OPTP0910
      II=I                    $      JI=I+1          $      NS=NSP(I)   OPTP0920
        DO 250 J=JI,NM1                                                 OPTP0930
        IF (NSP(J) .NE. NS) GO TO 250                                   OPTP0940
        IF (TRTY(J) .EQ. 1HC .OR. NNP(I) .NE. NNP(J)) GO TO 250         OPTP0950
C        COUNT THE SEGMENTS IN THE PIECE.                               OPTP0960
        L=1                                                             OPTP0970
          DO 210 K=JI,J                                                 OPTP0980
          IF (NSP(K) .NE. NSP(J+L)  .OR.  TRTY(K) .EC. 1HC  .OR.        OPTP0990
     1    TRTY(J+L) .EQ. 1HC) GO TO 220                                 OPTP1000
  210     L=L+1                                                         OPTP1010
C        LOOK FOR ONE WAY SEGMENTS ON THE PEICE TO BE REVERSED.         OPTP1020
  220   IF=I+L                $      JF=J-1                             OPTP1030
          DO 230 K=IF,JF                                                OPTP1040
          IF (ISTG(NWAY,NSP(K)) .EQ. 1) GO TO 250                       OPTP1050
  230     CONTINUE                                                      OPTP1060
        DN=0.                                                           OPTP1070
          DO 240 K=1,L                                                  OPTP1080
  240     DN=DN+STG(LEN,NSP(I+K-1))                                     OPTP1090
        IF (DN .LE. DO) GO TO 250                                       OPTP1100
C        SAVE LENGTH AND POINTERS                                       OPTP1110
        DO=DN       $      ISV=I          $      JSV=J        $    LSV=L OPTP1120
  250   CONTINUE                                                        OPTP1130
  260 CONTINUE                                                          OPTP1140
      IF (DO .EQ. 0.) GO TO 310                                         OPTP1150
                                                                        OPTP1160
      II=ISV      $      IF=II+LSV      $      JI=JSV    $     JF=JI+LSV OPTP1170
```

```
C         REVERSE PATH FROM NODE IF TC NODE JI                                     OPTP1180
  270 JI=JI-1         $       JF=JF-1                                              OPTP1190
      NSP(II)=NSP(JI)            $      TRTY(II)=TRTY(JI)                          OPTP1200
      II=II+1                    $      NNP(II)=NNP(JI)                            OPTP1210
      NSP(JF)=NSP(IF)            $      TRTY(JF)=TRTY(IF)                          OPTP1220
      IF=IF+1                    $      NNP(JF)=NNP(IF)                            OPTP1230
      IF (IF .LT. JI) GO TO 270                                                    OPTP1240
C         CLOSE UP PATH LIST                                                       OPTP1250
      L=2*LSV                                                                      OPTP1260
      DO 280 I=JF,NM1                                                              OPTP1270
      NSP(I-L)=NSP(I)            $      TRTY(I-L)=TRTY(I)                          OPTP1280
  280 NNP(I-L+1)=NNP(I+1)                                                          OPTP1290
      N=N-L                                                                        OPTP1300
      GO TO 200                                                                    OPTP1310
                                                                                   OPTP1320
C         CAUSE THE PATH TO END ON THE LAST COLLECTION SEGMENT.                    OPTP1330
  300 N=N-1                      $      NM1=N-1                                    OPTP1340
  310 IF (TRTY(NM1) .NE. 1HC) GO TO 300                                            OPTP1350
      RETURN                                                                       OPTP1360
      END                                                                          OPTP1370
```

```
      PROGRAM PHASE3(INPUT,OUTPUT,TAPE1,TAPE2,TAPE3,TAPE4,TAPE5=INPUT,  PHS30010
     1  TAPE7=0,TAPE9)                                                  PHS30020
                                                                        PHS30030
C        LATEST CHANGES --                                              PHS30040
C        JULY 16, 1976. HJI. ADDED DTF ARRAYS.                          PHS30050
C        JUNE 16, 1976. HJI. ADDED CALL TO CONNECT.  ADDED BUFFEROUT TO PHS30060
C        UNIT 7 TO FREE CORE TEMPORARILY.                               PHS30070
C        NOV 18, 1975. HJI.  ADDED SEARCH FOR ENTRY/EXIT POINTS         PHS30080
C        OCT 20, 1975. HJI. ADDED CONNECTIVITY CHECK.                   PHS30090
C        OCT 9, 1975. HJI.   ADDED PRINTING OF INPUT DATA.              PHS30100
C        OCT 3, 1975. HJI.   NSEG PUT IN BLANK COMMON. NEIGHBORING      PHS30110
C        SEGMENTS COUNTED AND LINE NUMBERS SAVED (ARRAYS NUMNBR AND     PHS30120
C        NBRSEG).                                                       PHS30130
                                                                        PHS30140
C        TAPE 1 IS SEGMENT DATA.                                        PHS30150
C        TAPE 2 IS NODE DATA.                                           PHS30160
C        TAPE 3 IS TRUCK DATA.                                          PHS30170
C        TAPE 4 IS THE SECTION LIST.                                    PHS30180
C        TAPE 9 IS THE FINAL ROUTE DESCRIPTION.                         PHS30190
                                                                        PHS30200
      COMMON NSEG,STG                                                   PHS30210
      COMMON /NODDATA/ KNODES,NODNUM(300),XNOD(300),YNOD(300),          PHS30220
     1  NUMNBR(300),NBRSEG(300)                                         PHS30230
      COMMON /TEMSTG/ SINL(50),SFNL(50),NNTF(300,3),DTF(300,3),         PHS30240
     1  TTF(300,3),NSTF(300,3),XXX,NTF(10,3),IPN(200),IPS(200),Z(4000), PHS30250
     2  IPATH(100),CORT(200),ISFS(100),ISTS(100),ISSV(200),NDFS(100),   PHS30260
     3  NDTS(100),NDSV(200),TRTY(200)                                   PHS30270
      COMMON /PART/ NEDGE,IPAIR(32)                                     PHS30280
      DIMENSION ISTG(11,500),STG(11,500),TC(50),TITLE(8),TLOAD(50)      PHS30290
      DIMENSION NNFD(1),NNFG(1),NNTD(1), TFD(1),TFG(1),TTD(1),          PHS30300
     1   NSFD(1),NSFG(1),NSTD(1),DFD(1),DFG(1),DTD(1)                   PHS30310
      DIMENSION NODORDS(100),NODPTRS(100),ONDMTX(1024)                  PHS30320
      DIMENSION DISTS(100), NN(2), NODORD(100),NODPTR(100),             PHS30330
     1  NFD(10), NFG(10), NTD(10)                                       PHS30340
      DIMENSION JNSG(200),JNN1(200),JNN2(200),JNWY(200),JTIM(200)       PHS30350
      DIMENSION LSRS(2)                                                 PHS30360
      EQUIVALENCE (STG,ISTG)                                            PHS30370
      EQUIVALENCE (NSTF,NSFD), (NSTF(1,2),NSFG), (NSTF(1,3),NSTD),      PHS30380
     1              (NNTF,NNFD), (NNTF(1,2),NNFG), (NNTF(1,3),NNTD),    PHS30390
```

```
2             ( TTF, TFD), ( TTF(1,2), TFG), ( TTF(1,3), TTD)         PHS30400
      EQUIVALENCE ( DTF, DFD), ( DTF(1,2), DFG), ( DTF(1,3), DTD)     PHS30410
      EQUIVALENCE (SINL,DISTS)                                        PHS30420
      EQUIVALENCE (NN1,NN(1)), (NN2,NN(2)), (NFD,NTF), (NFG,NTF(1,2)), PHS30430
     1   (NTD,NTF(1,3))                                               PHS30440
      DATA JMAX,KMAX/ 4,4/                                            PHS30450
      DATA LSRS/ 6HGARAGE, 8HLANDFILL/                                PHS30460
      DATA MAXODD/32/                                                 PHS30470
      DATA NNTF,NSTF,DTF,TTF,NUMNBR,NBRSEG/4200*0/                    PHS30480
      DATA NSTR,NN1,NN2,LEN,NH,NSPD,NWAY,NRQF,NSECT,MSECT,NSF/        PHS30490
     1        1,  2,  3,  4, 5,   6,   7,    8,    9,   10, 11/       PHS30500
      DATA TLOAD/50*0./                                               PHS30510
                                                                      PHS30520
      READ 5,TITLE                                                    PHS30530
    5 FORMAT(8A10)                                                    PHS30540
      READ (3) NSEG,NTRIP,(SINL(I),SFNL(I),TC(I),I=1,NTRIP)           PHS30550
      PRINT 10, TITLE                                                 PHS30560
   10 FORMAT(1H1,20X,8A10//*0INITIAL ASSIGNMENT OF SEGMENTS*)         PHS30570
      DO 16 I=1,NTRIP                                                 PHS30580
      JI=SINL(I)                  $       JF=SFNL(I)                  PHS30590
         DO 14 J=JI,JF                                                PHS30600
         READ (4,12) ISTG(1,J)                                        PHS30610
   12    FORMAT(I6)                                                   PHS30620
   14    ISTG(NSECT,J)=I                                              PHS30630
   16 PRINT 18, I,(ISTG(1,J),J=JI,JF)                                 PHS30640
   18 FORMAT(///*0SEGMENTS IN SECTION*,I4//(5X,30I4))                 PHS30650
                                                                      PHS30660
      CALL ISHLSRT(ISTG,ISTG(NSECT,1),JF,1,11)                        PHS30670
                                                                      PHS30680
      READ (1) NSEG,((STG(J,I),J=1,8),DUMMY,DUMMY,STG(NSF,I),I=1,NSEG), PHS30690
     1 AVMD                                                           PHS30700
      READ (2) NHTOT,TOTREF,KNODES,(NODNUM(I),NBRSEG(I),XNOD(I),YNOD(I),PHS30710
     1  I=1,KNODES)                                                   PHS30720
      DO 19 I=1,NSEG                                                  PHS30730
      IF (ISTG(NH,I) .EQ. 0) ISTG(NSECT,I)=0                          PHS30740
   19 ISTG(NH,I)=IABS(ISTG(NH,I))                                     PHS30750
                                                                      PHS30760
      DO 20 I=1,NSEG                                                  PHS30770
      J=ISTG(NSECT,I)                                                 PHS30780
```

```
20 IF (J .GT. 0) TLOAD(J)=TLOAD(J)+ISTG(NH,I)*STG(NRQF,I)                 PHS30790
                                                                          PHS30800
C     FIND THE NUMBER OF SEGMENTS MEETING AT EACH NODE                    PHS30810
   DO 36 I=1,KNODES                                                       PHS30820
   J=NBRSEG(I)              $      K=3                                    PHS30830
   IF (J .LT. 10000000000B) GO TO 34                                      PHS30840
   IF (J .GT. 200000000000000B) GO TO 32                                  PHS30850
   K=4                      $      GO TO 36                               PHS30860
32 K=6                                                                    PHS30870
   IF (J .LT. 40000000000000000B) K=5    $     GO TO 36                   PHS30880
34 IF (J .GT. 4000000B) GO TO 36                                          PHS30890
   K=2                      $      IF (J .LT. 2000B) K=1                  PHS30900
36 NUMNBR(I)=K                                                            PHS30910
                                                                          PHS30920
   READ 60, NOMP,NGAR                                                     PHS30930
   PRINT 40                                                               PHS30940
40 FORMAT(*1CHANGES TO SECTION ASSIGNMENTS*/ *0 SEGMENT   FROM           TPHS30950
  10       AMOUNT*/*  NUMBER   SECTION  SECTION      MOVED*//)            PHS30960
   CALL ADJUST(TLOAD,TC)                                                  PHS30970
   PRINT 45,(I,TLOAD(I),TC(I),I=1,NTRIP)                                  PHS30980
45 FORMAT(/40X,4(*  TRIP  LOAD  CAPACITY*)/ (40X,I6,2F8.2,I6,2F8.2,       PHS30990
  1 I6,2F8.2,I6,2F8.2))                                                   PHS31000
C     READ CHANGES TO SECTIONING                                          PHS31010
50 READ 60, NSGT,NSCN                                                     PHS31020
60 FORMAT(2I5)                                                            PHS31030
   IF (EOF(5)) 80,70                                                      PHS31040
70 NSCO=ISTG(NSECT,NSGT)   $      ISTG(NSECT,NSGT)=NSCN                   PHS31050
   IF (NSCN .LE. 0) GO TO 72                                              PHS31060
   RQ=ISTG(NH,NSGT)*STG(NRQF,NSGT)                                        PHS31070
   TLOAD(NSCN)=TLOAD(NSCN)+RQ  $      TLOAD(NSCO)=TLOAD(NSCO)-RQ          PHS31080
72 PRINT 75, NSGT,NSCO,NSCN,RQ                                            PHS31090
75 FORMAT(3I9,F9.2,*   (USER ADJUSTED)*)                                  PHS31100
   GO TO 50                                                               PHS31110
80 CONTINUE                                                               PHS31120
   PRINT 45,(I,TLOAD(I),TC(I),I=1,NTRIP)                                  PHS31130
   PRINT 82, (I,(STG(J,I),J=1,9),STG(11,I),I=1,NSEG)                      PHS31140
82 FORMAT(*1       NSTR  NN1  NN2  LEN   NH  SPD NWAY  RQF NSCT      SF*PHS31150
  1  /(4I5,F5.2,I5,F5.1,I5,F5.1,I5,O22))                                  PHS31160
                                                                          PHS31170
```

```
      CALL TREE(NOMP.OFD.TFD.0..NNFD,NSFD.1.KNODES.0)
      CALL TREE(NGAR.DFG.TFG.0..NNFG.NSFG.1.KNODES.0)
      CALL TREE(NOMP.DTD.TTD.0..NNTD,NSTD.-1.KNODES,0)

      CALL TRACE(NGAR,NOMP,NNFD,NSFD,IPS.IPN.NSG)
      CALL FLIP(IPS.NSG)      $      CALL FLIP(IPN,NSG+1)
      DIST=CUMDIS(IPS.NSG)
C        WRITE PATH FROM LAND FILL TO GARAGE ON TAPE9.
      WRITE (9.84) NSG.DIST
   84 FORMAT(I5.F10.3.I5.2F10.3)
      WRITE (9.85) (IPN(I),IPS(I),1HT,I=1,NSG),IPN(NSG+1)
   85 FORMAT(8(I5.I4.A1))

C        SAVE DATA ON DISK TO TEMPORARILY FREE CORE STORAGE
      IF (UNIT(7)) 86.86.86
   86 BUFFER OUT (7.1) (NNTF.XXX)
      IF (UNIT(7)) 87.87.87
   87 REWIND 7

      PRINT 90. NOMP.NGAR
   90 FORMAT(*1NODE*.I5.* IS THE DUMP,  NODE*,I5,* IS THE GARAGE*/*0LINE
     1  NODE   NNFD   NNTD   NNFG   NSFD  NSTD  NSFG    DFD    DTD    DFG
     2TFD     TTD     TFG    NUMNBR      NEIGHBORING SEGMENT NUMBERS*/)
      PRINT 100, (I.NODNUM(I),NNFD(I),NNTD(I),NNFG(I),NSFD(I),NSTD(I),
     1 NSFG(I).DFD(I),DTD(I),DFG(I),TFD(I).TTD(I).TFG(I),NUMNBR(I),
     2  (SHIFT(NBRSEG(I).10*(1-J)) .A. 1777B,J=1,6),I=1,KNODES)
  100 FORMAT(I5,7I6,6F7.3,I6.5X,6I5.0)

C        CONNECTIVITY CHECK
      CC=1H1
      DO 180 I=1,KNODES
      IPR=0                   $      NI=NODNUM(I)
      IF (NNFD(I) .NE. 0  .OR.  NI .EQ. NOMP) GO TO 140
      PRINT 130, CC.NI
  130 FORMAT(A1.*NODE*, I5,* DOES NOT CONNECT TO THE DUMP*)
      IPR=1                   $      CC=1H
  140 IF (NNFG(I) .NE. 0  .OR.  NI .EQ. NGAR) GO TO 180
      IF (IPR .GT. 0) GO TO 160
      PRINT 150, CC.NI
```

```
  150 FORMAT(A1,*NODE*,I5,* DOES NOT CONNECT TO THE GARAGE*)               PHS31570
      CC=1H                                                                PHS31580
      GO TO 180                                                            PHS31590
  160 PRINT 170                                                            PHS31600
  170 FORMAT(1H+,39X,*OR TO THE GARAGE*)                                   PHS31610
  180 CONTINUE                                                             PHS31620
      IF (CC .EQ. 1H1) GO TO 200                                           PHS31630
      PRINT 190                                                            PHS31640
  190 FORMAT(//*0CORRECT THE DUMP OR GARAGE NUMBERS IF INCORRECT*/* OR EPHS31650
     1LSE CORRECT THE MAP DESCRIPTION CARDS AND RERUN THE INPUT AND SECTPHS31660
     2IONING PROGRAMS */*0JOB TERMINATED*)                                 PHS31670
      CALL EXIT                                                            PHS31680
  200 CONTINUE                                                             PHS31690
                                                                           PHS31700
C        DO ROUTING FOR EACH SECTION                                       PHS31710
      DO 800 I=1,NTRIP                                                     PHS31720
         DO 202 J=1,NSEG                                                   PHS31730
  202    ISTG(MSECT,J)=ISTG(NSECT,J)                                       PHS31740
      CALL CONNECT(I,NPIECE)                                               PHS31750
         DO 210 J=1,100                                                    PHS31760
  210    NODORD(J)=NODPTR(J)=0                                             PHS31770
                                                                           PHS31780
C        COLLECT ALL NODES IN SECTION I                                    PHS31790
      KN=0                                                                 PHS31800
         DO 230 J=1,NSEG                                                   PHS31810
         IF (ISTG(MSECT,J) .NE. I) GO TO 230                               PHS31820
                                                                           PHS31830
C        SEGMENT J IS IN SECTION I.  SAVE BOTH ENDPOINTS.                  PHS31840
            DO 220 K=1,2                                                   PHS31850
            LINE=IFIND(ISTG(NN(K),J),NODNUM,KNODES)                        PHS31860
            NS=IFIND(LINE,NODPTR,KN)                                       PHS31870
            IF (NS .GT. 0) GO TO 220                                       PHS31880
                                                                           PHS31890
C        SAVE THE LINE NUMBER OF ENDPOINT NODNUM(LINE) IN NODPTR ARRAY     PHS31900
            NS=-NS                                                         PHS31910
            CALL MOVE3(NS,KN,NODPTR,NODORD,DISTS)                          PHS31920
            NODPTR(NS)=LINE                                                PHS31930
  220       NODORD(NS)=NODORD(NS)+1                                        PHS31940
  230    CONTINUE                                                          PHS31950
```

```
C        DELETE DANGLING SEGMENTS WHICH REQUIRE  NO COLLECTION
      CALL DISCON(I.KN,NODPTR,NODORD)
C        THERE ARE KN NODES IN SECTION I

      CALL EPXP(KN,NODPTR,NODORD,I)
C        SAVE NODORD AND NODPTR ARRAYS
       DO 240 J=1,KN
       NODORDS(J)=NODORD(J)
  240  NODPTRS(J)=NODPTR(J)
      KNS=KN

       DO 500 ITRIP=1,2
       SAVDIS=1.E20
       DO 470 J=1,JMAX
       ISTART=NTF(J,3-ITRIP)      $      IF (ISTART .EQ. 0) GO TO 480
        DO 460 K=1,KMAX
        ISTOP=NTF(K,3)            $     IF (ISTOP .EQ. 0) GO TO 470
C        RESTORE NODORD ANC NODPTR ARRAYS
          KN=KNS
            DO 250 KK=1,KN
            NODORD(KK)=NODORDS(KK)    $       NODPTR(KK)=NODPTRS(KK)
            N =NODNUM(NODPTR(KK))
            IF (N   .EQ. ISTART) NODORD(KK)=NODORD(KK)+1
  250       IF (N   .EQ. ISTOP)  NODORD(KK)=NODORD(KK)+1
C        SET NUMBER OF TRAVERSALS TO 1 FOR ALL SEGMENTS IN SECTION I.
            DO 260 KK=1,NSEG
            ISTG(NSF,KK)=0
  260       IF (ISTG(MSECT.KK)  .EQ. I) ISTG(NSF.KK)=1
C        CLOSE OFF ALL ORDER 1 NODES.
          CALL CLOSE1(I,KN,KNX,NODPTR,NODORD)
          KN=KNX
C        MOVE ODD NODES TO THE FRONT OF TABLES
          CALL MOVODD(KN,NODORD,NODPTR,NODD)
          IF (NODD .LE. MAXODD) GO TO 270
          PRINT 265, NODD,MAXODD,(NODNUM(NODPTR(KK)),NODORD(KK),KK=1,
     1     KN)
  265     FORMAT(*1THE NUMBER OF ODD NODES,*,I3,*, EXCEEDS*,I3/
     1    *OEITHER MODIFY THE MAP TO PAIR OR REMOVE ODD NODES OR ELSE*
     2    /* SEE THE PROGRAM DESCRIPTION MANUAL FOR INSTRUCTIONS*/
```

```
    3       *ONODE/ORDER*/ 10(I6,1H/,I1))                               PHS32350
            CALL PRNPCS(I,1)         $       GO TO 460                  PHS32360
  270       CONTINUE                                                    PHS32370
C        GENERATE DISTANCE MATRIX FOR ODD NODES                         PHS32380
              CALL GENDM(ONDMTX,NODD,NODPTR,I)                          PHS32390
C        GENERATE AN INITIAL PAIRING OF ODD NODES                       PHS32400
            CALL SELORD(NODD.NODD,ONDMTX.IPAIR,-1)                      PHS32410
C        MINIMIZE TOTAL PAIRING DISTANCE                                PHS32420
            IERR=0                                                      PHS32430
            IF (NODD .GT. 2) CALL SOLV(ONDMTX,NODD,IERR)                PHS32440
            IF (IERR .GT. 0) GO TO 460                                  PHS32450
C        DOUBLE THE SEGMENTS WHICH CONNECT THE ODD NODE PAIRS.          PHS32460
              DO 290 KK=1,NODD                                          PHS32470
              IF (KK  .GT. IPAIR(KK)) GO TO 290                         PHS32480
              N1=NODPTR(KK)   $     N2=NODPTR(IPAIR(KK))                PHS32490
C        FIND THE SHORTEST PATH FROM NODNUM(N1) TO NODNUM(N2) USING ONLYPHS32500
C        SEGMENTS IN SECTION I.                                         PHS32510
                                                                        PHS32520
                DO 260 L=1,KNODES                                       PHS32530
                DTF(L,1)=TTF(L,1)=0.                                    PHS32540
  280           NNTF(L,1)=NSTF(L,1)=0                                   PHS32550
              CALL TREE(NODNUM(N2),DTF,TTF,1.,NNTF,NSTF,1,100,I)        PHS32560
              JJ=0                                                      PHS32570
  285         M =NSTF(N1.1)   $     ISTG(NSF, M)=ISTG(NSF, M)+1         PHS32580
              N1=NNTF(N1.1)                                             PHS32590
              JJ=JJ+1         $     IPATH(JJ)=M                         PHS32600
              IF (N1 .NE. N2 .AND. N1 .NE. 0) GO TO 285                 PHS32610
  290         CONTINUE                                                  PHS32620
                                                                        PHS32630
C        IF ANY TWO WAY STREET IS TRAVELED MORE THAN TWICE REMOVE PAIRS PHS32640
C        OF TRAVERSALS.                                                 PHS32650
            DO 300 L=1.NSEG                                             PHS32660
            IF (ISTG(NSF.L) .LE. 2) GO TO 300                           PHS32670
            IF (ISTG(NWAY,L) .NE. 1) ISTG(NSF,L)=MOD(ISTG(NSF,L)-1,2)+1 PHS32680
  300       CONTINUE                                                    PHS32690
              DO 310 L=1.100                                            PHS32700
  310         IPN(L)=IPS(L)=0                                           PHS32710
            JSEG=0                                                      PHS32720
              DO 380 L=1.NSEG                                           PHS32730
```

```
      IF (ISTG(MSECT,L) .NE. I .AND. ISTG(NSF,L) .EQ. 0) GO TO         PHS32740
     1   380                                                           PHS32750
      JSEG=JSEG+1                 $    JNSG(JSEG)=L                    PHS32760
      JNN1(JSEG)=ISTG(NN1,L)      $    JNN2(JSEG)=ISTG(NN2,L)          PHS32770
      JTIM(JSEG)=ISTG(NSF,L)      $    JNWY(JSEG)=ISTG(NWAY,L)         PHS32780
380   CONTINUE                                                         PHS32790
      CALL TRAVEL(ISTART,ISTOP,JSEG,JNN1,JNN2,JNSG,JNWY,JTIM,M,        PHS32800
     1IPN,IPS,IERR)                                                    PHS32810
      IF (IERR .NE. 0) PRINT 381                                       PHS32820
381   FORMAT(*0TRAVEL PATH CANNOT BE FOUND. RECHECK NETWORK FOR THPHS32830
     1IS TRIP*)                                                        PHS32840
      MM1=M-1                                                          PHS32850
      IF (IERR .EQ. 0) CALL OPTPATH(IPN,IPS,TRTY,M,I)                  PHS32860
      ISTART=IPN(1)     $      ISTOP=IPN(M)     $      MM1=M-1         PHS32870
      IF (UNIT(7)) 390,390,390                                         PHS32880
390   BUFFER IN (7,1) (NNTF,XXX)                                       PHS32890
      IF (UNIT(7)) 400,400,400                                         PHS32900
400   REWIND 7                                                         PHS32910
      IF (IERR .NE. 0) GO TO 460                                       PHS32920
      IF (ITRIP .EQ. 2) GO TO 410                                      PHS32930
      CALL TRACE(ISTART,NGAR,NNFG,NSFG,ISTS,NDTS,NSTS)                 PHS32940
      GO TO 420                                                        PHS32950
410   CALL TRACE(ISTART,NDMP,NNFD,NSFD,ISTS,NDTS,NSTS)                 PHS32960
420   CALL FLIP(ISTS,NSTS)   $     CALL FLIP(NDTS,NSTS+1)              PHS32970
      CALL TRACE(ISTOP,NDMP,NNTD,NSTD,ISFS,NDFS,NSFS)                  PHS32980
      D1=CUMDIS(ISTS,NSTS)   $     D2=CUMDIS(IPS,MM1)                  PHS32990
      D3=CUMDIS(ISFS,NSFS)   $     DIST=D1+D2+D3                       PHS33000
      IF (DIST .GE. SAVDIS) GO TO 460                                  PHS33010
      SAVDIS=DIST                                                      PHS33020
        DO 430 KK=1,NSTS                                               PHS33030
        ISSV(KK)=ISTS(KK)                                              PHS33040
        CORT(KK)=1HT                                                   PHS33050
430     NDSV(KK)=NDTS(KK)                                              PHS33060
        DO 440 KK=1,MM1                                                PHS33070
        ISSV(KK+NSTS)=L=IPS(KK)                                        PHS33080
        NDSV(KK+NSTS)=IPN(KK)                                          PHS33090
440     CORT(KK+NSTS)=TRTY(KK)                                         PHS33100
        DO 450 KK=1,NSFS                                               PHS33110
        ISSV(KK+NSTS+MM1)=ISFS(KK)                                     PHS33120
```

```
            NDSV(KK+NSTS+MM1)=NDFS(KK)                                  PHS33130
450         CORT(KK+NSTS+MM1)=1HT                                       PHS33140
          NDSV(NSTS+M+NSFS)=NDMP                                        PHS33150
          NSTSSV=NSTS        $     NSISSV=MM1     $     NSFSSV=NSFS     PHS33160
          SD1=D1             $     SD2=D2         $     SD3=D3          PHS33170
460       CONTINUE                                                      PHS33180
470     CONTINUE                                                        PHS33190
480     IF (SAVDIS .LT. 1.E19) GO TO 490                                PHS33200
        PRINT 485, LSRS(ITRIP)                                          PHS33210
485     FORMAT(*0NO PATH EXISTS FOR THE TRIP STARTING AT THE *,A10/)    PHS33220
        GO TO 500                                                       PHS33230
490     WRITE (9,84)  NSTSSV,SD1,I,TC(I),TLOAD(I)                       PHS33240
        WRITE (9,85)  (NDSV(J),ISSV(J),CORT(J),J=1,NSTSSV),             PHS33250
     1                NDSV(NSTSSV+1)                                    PHS33260
        JI=NSTSSV+1          $     JF=JI+NSISSV-1                       PHS33270
        WRITE (9,84)  NSISSV,SD2,I,TC(I),TLOAD(I)                       PHS33280
        WRITE (9,85)  (NDSV(J),ISSV(J),CORT(J),J=JI,JF),NDSV(JF+1)      PHS33290
        JI=JF+1              $     JF=JI+NSFSSV-1                       PHS33300
        WRITE (9,84)  NSFSSV,SD3,I,TC(I),TLOAD(I)                       PHS33310
        WRITE (9,85)  (NDSV(J),ISSV(J),CORT(J),J=JI,JF),NDSV(JF+1)      PHS33320
500   CONTINUE                                                          PHS33330
800   CONTINUE                                                          PHS33340
      ENDFILE 9                                                         PHS33350
      CALL EXIT                                                         PHS33360
      END                                                               PHS33370
```


(The reverse of this page is blank.)

APPENDIX C
DEFINITIONS OF IMPORTANT VARIABLES

Note: A single variable symbol may have different meanings in relation to the various subroutines. For this reason, variables are defined below for each subroutine and for program PHASE3.

SUBROUTINE FLIP

N	A count of the items to be reversed.
X	The array to be reversed.

SUBROUTINE MOVE3

A1, A2, A3	Arrays to be moved.
IF	Subscript data are moved to.
II	Subscript data come from.

FUNCTION IFIND

IARRAY	Array being searched.
LEN	Length of IARRAY.
NUM	Number being sought.

SUBROUTINE SHLSRT3

A, B	Arrays reordered as array X is sorted.
NW	Count of words to be sorted.
SGN	If SGN = 1.0, X is sorted into increasing order. If SGN = -1.0, X is sorted into decreasing order.
X	Array to be sorted.

SUBROUTINE ISHLSRT

IA	Array reordered as array IX is sorted.
INC	The spacing between words to be sorted.
ISGN	If ISGN = 1, IX is sorted into increasing order. If ISGN = -1, IX is sorted into decreasing order.

IX	Array to be sorted.
NW	Count of words to be sorted.

SUBROUTINE ADJUST

ISTG	Array of integer-valued segment data.
KNODES	Count of nodes.
NBRSEG	Array of packed neighboring-segment numbers.
NH	Symbolic subscript of ISTG array for house count.
NRQF	Symbolic subscript of STG array for refuse-quantity adjustment factor.
NSC1, NSC2	Section assignment numbers.
NSECT	Symbolic subscript of ISTG array for section assignment.
NSG1, NSG2	Segment numbers.
R1, R2	Refuse quantities.
STG	Arrays of floating point-valued segment data.
TC	Array of vehicle capacities.
TL	Array of vehicle loads.

FUNCTION CUMDIS

ISEG	Array of segment numbers.
ISTG	Array of integer-valued segment data.
NSG	Count of segments.
STG	Array of floating point-valued segment data.

SUBROUTINE PRNPCS

ISTG	Array of integer-valued segment data.
ITRIP	Piece number.
MSECT	Symbolic subscript of ISTG array for temporary section assignment.
NPIECE	Count of pieces whose segments will be printed.
NSEG	Count of segments in map description.
NTRIP	Section number.

SUBROUTINE TRACE

IPN	Array to be filled with node numbers in path.
IPS	Array to be filled with segment numbers in path.
KNODES	Count of nodes in map description.
LEND	Line number of terminal node of path.
LPREVN	Array of line numbers of next node in path.
NODNUM	Array of node numbers.
NPREVS	Array of numbers of next segment in path.
NSG	Count of segments in path.
NSTART	Node number of start of path.
NSTOP	Node number of end of path.

SUBROUTINE MOVODD

KN	Count of nodes in section.
NODD	Count of odd nodes in section.
NODORD	Array of orders of nodes in section.
NODPTR	Array of pointers to nodes in section.

SUBROUTINE TREE

CUM	The weighted time and distance value from a node to the starting node.
DIST	An array of distances from the nodes to the starting node.
INSECT	If zero, all segments may be used; otherwise, only segments in section INSECT are used.
IST	An array of four pointers for each element in the tree (node pointer, left-branch pointer, right-branch pointer, and back pointer).
ISTG	Array of integer-valued segment data.
KNODES	Count of nodes in map description.
LA	Back-pointer value.
MAXLVL	Maximum number of steps allowed away from the starting node.
NEWS	Variable that indicates whether any new nodes have been added to the tree.
NEXT	Line number of next available free storage in the IST array.

NIU	Array of weighted time and distance values from nodes to the starting node.
NIULOC	NIULOC(I) gives the line number in the IST array of node NODNUM(I).
NODNUM	Array of node numbers.
NPREVN	Array of line numbers of next node in path to starting node.
NPREVS	Array of segment numbers of next segment in path to starting node.
STG	Array of floating point-valued segment data.
TIME	Array of travel times from starting node to each node.
W	Coefficient of distance in weighted time and distance value.

SUBROUTINE CON2ST

DIST	Length of two-segment path.
ISEG	Number of first segment in two-segment path.
ISTG	Array of integer-valued segment data.
JSEG	Number of second segment in two-segment path.
KNODES	Count of nodes in map description.
MSECT	Symbolic subscript of ISTG array for temporary section assignment.
NBRSEG	Array of packed neighboring-segment numbers.
NCON	Count of pieces connected by two-step paths.
NP1, NP2	Arrays of piece numbers.
NS	Count of entries in NP1 and NP2 arrays.
NSEG	Count of segments in map description.
NS1, NS2	Arrays of segment numbers of segments in two-step paths.
NTRIP	Section number.
NUMNBR	Array of counts of neighboring segments.
STG	Array of floating point-valued segment data.

SUBROUTINE FNDPTH

IP	Number of piece containing node NORG.
IPATH	Double-subscripted array to be filled with segment numbers of up to five paths connecting pieces of section NTRIP to node NORG.

ISTG	Array of integer-valued segment data.
KNODES	Count of nodes in map description.
LNF	Array of line numbers of nodes starting paths to node NORG.
LPREVN	Array of line numbers of next node in path.
NDSV	Count of paths saved in array IPATH.
NODNUM	Array of node numbers.
NPF	Array of piece numbers from which paths start.
NPREVS	Array of numbers of next segment in path.
NTRIP	Section number.
TIM	Array of travel times for paths.

SUBROUTINE CONNST

IPATH	Two-dimensional array of segment numbers in paths to node NORG.
ISTG	Array of integer-valued segment data.
LNF	Array of line numbers of nodes from which paths in array IPATH start.
MSECT	Symbolic subscript of ISTG array for temporary section assignment.
NDSV	Count of paths ending at node NORG.
NORG	The number of a node in the current section.
NPF	An array of piece numbers from which paths to node NORG start.
NPIECE	Count of pieces of section NTRIP.
NSEG	Count of segments in map description.
NTRIP	Section number.
TIM	Array of travel times for paths in array IPATH.

SUBROUTINE CONNECT

ISTG	Array of integer-valued segment data.
ITRIP	Piece number.
KNODES	Count of nodes in map description.
MSECT	Symbolic subscript of ISTG array for piece number.
NTRIP	Section number.

SUBROUTINE DISCON

ISTG	Array of integer-valued segment data.
KN	Count of nodes in section.
NBRSEG	Array of packed neighboring-segment numbers.
NODNUM	Array of node numbers.
NODORD	Array of orders of nodes in the current section.
NODPTR	Array of pointers to nodes in current section.
NTRIP	Section number.

SUBROUTINE EPXP

DISTS	Array of travel times to nodes from the landfill or garage.
I	Section number.
KLIM	Maximum number of entry- or exit-point nodes to be saved.
KN	Count of nodes in section I.
NNTF	Two-dimensional array of line numbers of the next node in the path to or from the landfill or garage.
NODORD	Array of orders of nodes in the section.
NODPTR	Array of pointers to nodes in the section.

SUBROUTINE CLOSE1

DIST	Array of distances from nodes to the order-one node.
IDIR	IDIR = 1 for travel away from the starting node or -1 for travel toward the starting node.
IPATHN	Array of line numbers of nodes in the path from the section to the order-one node.
IPATHS	Array of segment numbers in the path from the section to the order-one node.
ISTG	Array of integer-valued segment data.
KN	Count of nodes in the section before CLOSE1 is called.
KNODES	Count of nodes in the map description.
KNX	Count of nodes in the section after CLOSE1 is called.
LPREVN	Array of line numbers of the next or previous node in the path to or from the order-one node.
MAXLVL	The maximum number of steps allowed in the path from the order-one node to the section.

MSECT	Symbolic subscript of the ISTG array for temporary section assignment.
NBRS	Word of packed neighboring-segment numbers.
NODNUM	Array of node numbers.
NODORD	Array of orders of nodes in the section.
NODPTR	Array of pointers to nodes in the section.
NORG	The node number of an order-one node.
NOTHER	Number of the node at the other end of the segment from the order-one node.
NPREVS	Array of numbers of the next or previous segment in the path to or from the order-one node.
NPSV	Count of segments in the path from an order-one node to the section.
NSF	Symbolic subscript of ISTG array for number of times the segment is traversed.
NSSV	Count of segments saved in IPSSV array.
NTRIP	Section number.
TIME	Array of travel times between nodes and the order-one node.

SUBROUTINE GENDM

DIST	Array of distances from node NORG.
KNODES	Count of nodes in map description.
NN	Array of line numbers of next node in path to node NORG.
NODD	Count of odd nodes.
NODNUM	Array of node numbers.
NODPTR	Array of pointers to node numbers.
NORG	Number of node used as the start of a minimum-distance tree.
NS	Array of the number of the next segment in the path to node NORG.
NTRIP	Section number.
ONDMTX	Matrix of distances between odd nodes.
TIME	Array of travel times from nodes to node NORG.

SUBROUTINE SELORD

DIST	Matrix of distances between odd nodes.
IDIR	If IDIR = 1, the nodes with the smallest near-neighbor distance are paired first; if IDIR = -1, the nodes with the largest near-neighbor distance are paired first.
IPART	Array of sequence numbers of other nodes in odd-node pairs.
NNN	Dimension of distance matrix.
NODD	Count of odd nodes.

SUBROUTINE PATH

II	Sequence number of starting node of first path.
IPART	Array of sequence numbers of other nodes in odd-node pairs.
ISTP	Array of sequence numbers of nodes in path.
ISTPN	Matrix of packed sequence numbers of nodes in new path.
ISTPO	Matrix of packed sequence numbers of nodes in old path.
JJ	Sequence number of node ending second path.
KK	Sequence number of node ending first path and starting second path.
MID	Count of nodes through the end of the first path.
NEDGE	Count of odd nodes.
NG	Error indicator returned by subroutine PATH.
NNPW	Maximum number of sequence numbers that can be packed into one computer word.

SUBROUTINE NEXTM

ISTP	Array of sequence numbers of nodes in the current path.
ISTPN	Matrix of sequence numbers of nodes in the new paths between odd nodes.
ISTPO	Matrix of sequence numbers of nodes in old paths between odd nodes.
JSTP	Array of sequence numbers of nodes in the most profitable cycle.
MTXN	Matrix of profits in new paths between odd nodes.
MTXO	Matrix of profits in old paths between odd nodes.
NEDGE	Count of odd nodes.
NG	Error indicator.
NRES	Profitable-cycle indicator.

SUBROUTINE SOLV

DIST	Matrix of distances between odd nodes.
IPART	Array of sequence numbers of other odd nodes in odd-node pairs.
ISTP1, ISTP2	Matrices of packed sequence numbers of nodes in paths between odd nodes.
JSTP	Array of sequence numbers of odd nodes in profitable cycle.
MTX1, MTX2	Matrices of profits in paths between odd nodes.
NCH	Iteration count.
NNPW	Maximum number of node sequence numbers that can be packed into a word.
NODD	Count of odd nodes.
NRES	Profitable-cycle indicator.

SUBROUTINE TRAVEL

IERR	Error indicator.
IPN	Array of numbers of nodes in path.
IPS	Array of numbers of segments in path.
ISP	Number of node at which path ends.
IST	Number of node at which path starts.
KLINE	Array of line numbers of segments in path.
KS	Array of backup counts.
KTIMES	Array of counts of times segments are available for traversal.
MAXKS	Maximum backup count allowed before path search is terminated.
NEXTII	Array of line number to be used next in adding a segment to a particular step of the path.
NN	Count of nodes in path.
NN1, NN2	Arrays of numbers of nodes starting and ending the segments.
NSEG	Count of segments in section.
NSG	Array of segment numbers.
NTIMES	Array of number of times each segment must be used.
NWAY	Array giving the number of ways a segment may be traversed.

SUBROUTINE OPTPATH

DN	Length of new travel stretch.
DO	Length of old travel stretch.
IDEL	Difference between new and old counts of steps in travel stretches.
IS	Count of steps in old travel stretch.
ISC	Section number.
ISTG	Array of integer-valued segment data.
LEN	Symbolic subscript of STG array for segment length.
NH	Symbolic subscript of ISTG array for house count.
NNP	Array of node numbers in path.
NS	Count of segments in new travel stretch.
NSC	Symbolic subscript of ISTG array for section assignment.
NSP	Array of numbers of segments in path.
STG	Array of floating point-valued segment data.
TRTY	Array of travel-or-collection indicators for each segment in path.

PROGRAM PHASE3

CORT	Array of travel-or-collection indicators for path saved for output.
DFD	Array of distances from landfill.
DFG	Array of distances from garage.
DIST	Total length of a trip.
DTD	Array of distances to landfill.
DTF	Matrix of distances to and from garage and landfill for entry- and exit-point nodes.
IERR	Error indicator.
IPAIR	Array of sequence numbers of other nodes in odd-node pairs.
IPN	Array of numbers of nodes in travel path.
IPS	Array of numbers of segments in travel path.
ISFS	Array of numbers of segments in path from section to landfill.
ISSV	Array of numbers of segments saved for output in final travel path.
ISTART	Number of entry-point node.
ISTG	Array of integer-valued segment data.

ISTOP	Number of exit-point node.
ISTS	Array of numbers of segments in path from garage to section.
JMAX	Maximum number of entry-point nodes to be used.
KMAX	Maximum number of exit-point nodes to be used.
KN	Count of nodes in section.
KNODES	Count of nodes in map description.
MAXODD	Maximum number of odd nodes that can be paired.
MSECT	Symbolic subscript of ISTG array for temporary section assignment.
NBRSEG	Array of packed bounding-segment numbers.
NDFS	Array of numbers of nodes in path from section to landfill.
NDMP	Node number of the landfill.
NDSV	Array of numbers of nodes in travel path saved for output.
NDTS	Array of numbers of nodes in path from garage to section.
NGAR	Node number of the garage.
NH	Symbolic subscript of array ISTG for house count.
NN1, NN2	Symbolic subscripts of ISTG array for starting and ending node numbers.
NODD	Count of odd nodes.
NODNUM	Array of node numbers.
NODORD	Array of orders of nodes in section.
NODPTR	Array of pointers to nodes in section.
NSECT	Symbolic subscript of ISTG array for permanent section assignment.
NSEG	Count of segments in map description.
NTRIP	Count of sections.
NUMNBR	Array of counts of neighboring segments.
ONDMTX	Matrix of distances between odd nodes.
STG	Array of floating point-valued segment data.
TC	Array of vehicle capacities.
TLOAD	Array of vehicle loads.
TRTY	Array of travel-or-collection indicators for current travel path.

APPENDIX D

SAMPLE PRINTED OUTPUT

KIRTLAND AFB (EAST), NM

INITIAL ASSIGNMENT OF SEGMENTS

SEGMENTS IN SECTION 1

1 2 3 149 18 61 17 4 150 68 66 60 5 64 59 35 34 36 69

SEGMENTS IN SECTION 2

16 67 65 72 27 26 25 15 47 48 33 32 24 56 23 49 55 31 14

SEGMENTS IN SECTION 3

57 38 22 52 30 37 21 29 39 9 28 58 51 73 20 13 50 19 10

SEGMENTS IN SECTION 4

11 70 12 8 41 40 43 7 42 53 44 54 45 62 5 74 71 168 75 81 76 46 63 82 84 169 83 77 159 78
144 85 120 118 145 152 160 153 154 155 167 79 119 117 121 146 114 116 110 80 111 122 112

SEGMENTS IN SECTION 5

147 123 102 100 99 124 101 140 141 139 125 103 137 90 127 91 92 89 94 135 93 95 134 142 136 96 113 98 143 97
133 148 161 104 126 138 88 105 132 87 131 86 130 129 109 108 106 128 157 156 107 233 158 231

SEGMENTS IN SECTION 6

205 206 204 232 203 207 202 234 170 208 209 235 239 201 200 230 244 199 240 229 198 241 197 246 228 196 210 171 226 236
211 245 172 173 195 174 237

SEGMENTS IN SECTION 7

183 166 184 247 165 185 164 175 248 163 176 162 186 187 177 189 178 180 188 179

SEGMENTS IN SECTION 8

181 182 190 191 212 238 192 213 214 215 225 216 115 151 193 194 217 218 219 220 221 222 223 224 227 242 243 249 250

CHANGES TO SECTION ASSIGNMENTS

SEGMENT NUMBER	FROM SECTION	TO SECTION	AMOUNT MOVED	
12	4	3	6.00	(PROGRAM ADJUSTED)
27	2	1	5.00	(PROGRAM ADJUSTED)
228	6	8	13.00	(PROGRAM ADJUSTED)
231	5	6	12.00	(PROGRAM ADJUSTED)
234	6	5	15.00	(PROGRAM ADJUSTED)
239	8	6	12.00	(PROGRAM ADJUSTED)

TRIP	LOAD	CAPACITY	TRIP	LOAD	CAPACITY	TRIP	LOAD	CAPACITY	TRIP	LOAD	CAPACITY
1	213.00	220.00	2	208.00	220.00	3	213.00	220.00	4	205.00	220.00
5	216.00	220.00	6	207.00	220.00	7	192.00	220.00	8	192.00	220.00

SEGMENT NUMBER	FROM SECTION	TO SECTION	AMOUNT MOVED	
233	5	6	12.00	(USER ADJUSTED)
234	5	6	15.00	(USER ADJUSTED)
115	8	4	22.00	(USER ADJUSTED)
237	6	8	13.00	(USER ADJUSTED)
239	6	8	12.00	(USER ADJUSTED)
214	8	6	5.00	(USER ADJUSTED)
226	6	8	17.00	(USER ADJUSTED)
212	8	6	3.00	(USER ADJUSTED)
215	8	6	8.00	(USER ADJUSTED)
144	4	5	2.00	(USER ADJUSTED)
145	4	5	4.00	(USER ADJUSTED)
146	4	5	4.00	(USER ADJUSTED)
116	4	5	9.00	(USER ADJUSTED)
27	1	2	5.00	(USER ADJUSTED)
69	1	2	26.00	(USER ADJUSTED)
94	1	2	30.00	(USER ADJUSTED)
63	4	1	5.00	(USER ADJUSTED)
46	4	1	31.00	(USER ADJUSTED)
33	2	1	4.00	(USER ADJUSTED)
44	4	1	2.00	(USER ADJUSTED)
45	4	1	3.00	(USER ADJUSTED)
14	2	3	19.00	(USER ADJUSTED)
56	2	3	19.00	(USER ADJUSTED)
55	2	4	14.00	(USER ADJUSTED)
22	3	2	4.00	(USER ADJUSTED)
30	3	2	4.00	(USER ADJUSTED)
11	4	3	17.00	(USER ADJUSTED)
9	3	4	17.00	(USER ADJUSTED)
39	3	4	15.00	(USER ADJUSTED)
29	3	4	8.00	(USER ADJUSTED)
29	3	4	4.00	(USER ADJUSTED)

TRIP	LOAD	CAPACITY	TRIP	LOAD	CAPACITY	TRIP	LOAD	CAPACITY	TRIP	LOAD	CAPACITY
1	203.00	220.00	2	207.00	220.00	3	216.00	220.00	4	208.00	220.00
5	208.00	220.00	6	208.00	220.00	7	192.00	220.00	8	204.00	220.00

	NSTR	NN1	NN2	LEN	NH	SPD	NWAY	RQF	NSCT	SF
1	1	10	20	.96	0	30.0	2	1.0	0	00000000000000000000
2	2	20	30	.27	0	30.0	2	1.0	0	00000000000000000000
3	2	30	40	.32	0	30.0	2	1.0	0	00000000000000000000
4	2	40	50	.32	0	30.0	2	1.0	0	60623656050753412172
5	3	140	60	.23	34	15.0	2	1.0	1	00000000000000000000
6	3	60	70	.09	10	15.0	2	1.0	4	00000000000000000000
7	3	70	80	.03	3	15.0	2	1.0	4	00000000000000000000
8	4	80	90	.12	16	15.0	2	1.0	4	00000000000000000000
9	4	90	100	.13	17	15.0	2	1.0	4	00000000000000000000
10	4	100	110	.05	5	15.0	2	1.0	3	00000000000000000000
11	4	110	120	.12	17	15.0	2	1.0	3	60662702436560507537
12	4	120	130	.05	6	15.0	2	1.0	3	00000000000000000000
13	5	130	140	.15	20	15.0	2	1.0	3	00000000000000000000
14	5	140	150	.15	19	15.0	2	1.0	3	00000000000000000000
15	5	150	160	.19	24	15.0	2	1.0	2	00000000000000000000
16	6	160	170	.19	28	15.0	2	1.0	2	00000000000000000000
17	6	170	180	.22	31	15.0	2	1.0	1	00000000000000001423
18	6	180	190	.05	5	15.0	2	1.0	1	00000000000000000000
19	9	120	270	.05	5	15.0	2	1.0	3	00000000000000000000
20	9	270	280	.05	6	15.0	2	1.0	3	00000000000000000000
21	9	280	200	.05	5	15.0	2	1.0	3	00000000000000000000
22	9	200	290	.05	4	15.0	2	1.0	2	00000000000000000000
23	9	290	300	.05	6	15.0	2	1.0	2	00000000000000000000
24	9	300	250	.05	5	15.0	2	1.0	2	00000000000000000000
25	9	250	310	.05	5	15.0	2	1.0	2	00000000000000000000
26	9	310	320	.05	5	15.0	2	1.0	2	00000000000000000000
27	9	320	330	.05	5	15.0	2	1.0	2	00000000000000000000
28	13	100	360	.07	6	15.0	2	1.0	4	00000000000000000000
29	13	360	210	.06	4	15.0	2	1.0	4	00000000000000000000
30	13	210	400	.05	4	15.0	2	1.0	2	00000000000000000000
31	13	400	390	.05	4	15.0	2	1.0	2	00000000000000000000
32	13	390	240	.05	6	15.0	2	1.0	2	00000000000000000000
33	13	240	410	.05	4	15.0	2	1.0	1	00000000000000000000
34	13	410	420	.05	4	15.0	2	1.0	1	00000000000000000000
35	13	420	430	.05	4	15.0	2	1.0	1	00000000000000000000
36	13	430	170	.05	4	15.0	2	1.0	1	00000000000000000000
37	7	140	200	.05	3	15.0	2	1.0	3	00000000000000000000
38	7	200	210	.15	19	15.0	2	1.0	3	00000000000000000000
39	7	210	450	.11	15	15.0	2	1.0	4	00000000000000000000
40	7	450	470	.03	2	15.0	2	1.0	4	00000000000000000000
41	7	470	220	.07	10	15.0	2	1.0	4	00000000000000000000
42	7	220	70	.05	5	15.0	2	1.0	4	00000000000000000000
43	7	450	470	.06	5	15.0	2	1.0	4	00000000000000001423
44	8	50	50	.04	2	15.0	2	1.0	1	00000000000000000000
45	8	60	230	.05	3	15.0	2	1.0	1	00000000000000000000
46	8	230	240	.23	31	15.0	2	1.0	1	17156050753412172702
47	8	240	250	.15	19	15.0	2	1.0	2	00000000000000000000
48	8	250	150	.05	4	15.0	2	1.0	2	00000000000000000000
49	8	150	260	.04	2	15.0	2	1.0	2	00000000000000000000
50	10	110	270	.15	20	15.0	2	1.0	3	17135075341217270244
51	11	90	360	.15	20	15.0	2	1.0	3	17135075341217270244
52	11	360	240	.15	20	15.0	2	1.0	3	00000000000000000000
53	12	220	370	.06	5	15.0	2	1.0	4	00000000000000000000
54	12	370	380	.09	14	15.0	2	1.0	4	00000000000000000000
55	12	380	390	.11	14	15.0	2	1.0	4	17145075341217270244
56	12	390	300	.15	19	15.0	2	1.0	3	00000000000000000000
57	14	290	400	.15	18	15.0	2	1.0	3	00000000000000000000
58	14	400	380	.12	14	15.0	2	1.0	3	17145605075341217272
59	15	480	500	.07	8	15.0	2	1.0	1	17125075341217270244
60	15	500	510	.05	5	15.0	2	1.0	1	00000000000000000000
61	15	510	180	.04	5	15.0	2	1.0	1	00000000000000000000
62	16	370	230	.05	4	15.0	2	1.0	4	00000000000000002203
63	16	230	480	.06	5	15.0	2	1.0	1	00000000000000002203

NODE 13 IS THE DUMP. NODE 1160 IS THE GARAGE

LINE	NODE	NNFD	NNTD	NNFG	NSFD	NSTD	NSFG	DFD	DTD	DFG	TFD	TTD	TFG	NUMNBR	NEIGHBORING SEGMENT NUMBERS			
1	13	0	0	2	0	0	1	0.000	0.000	2.100	0.000	0.000	4.200	1	1			
2	23	1	1	3	1	1	2	.960	.960	1.140	1.920	1.920	2.200	2	2	1		
3	33	2	2	113	2	2	149	1.230	1.230	.870	2.460	2.460	1.740	3	149	3	2	
4	43	3	3	3	3	3	3	1.550	1.550	1.190	3.100	3.100	2.380	2	4	3		
5	53	4	4	4	4	4	4	1.870	1.870	1.510	3.740	3.740	3.020	2	44	4		
6	63	5	5	5	44	44	44	1.910	1.910	1.550	3.900	3.900	3.180	4	45	44	6	5
7	73	6	6	6	6	6	6	2.000	2.000	1.640	4.260	4.260	3.540	3	42	7	6	
8	83	7	7	7	7	7	7	2.030	2.030	1.670	4.380	4.380	3.660	2	8	7		
9	93	8	8	34	8	8	51	2.150	2.150	1.310	4.860	4.860	3.952	3	51	9	8	
10	103	9	9	34	9	9	28	2.280	2.280	1.230	5.380	5.380	3.632	4	70	28	10	9
11	113	10	10	27	10	10	50	2.330	2.330	1.210	5.580	5.580	3.552	3	50	11	10	
12	123	27	27	27	19	19	19	2.530	2.530	1.110	5.992	5.992	3.152	3	19	12	11	
13	133	14	14	14	13	13	13	2.480	2.480	1.060	5.792	5.792	2.952	2	13	12		
14	143	15	15	15	14	14	14	2.330	2.330	.910	5.192	5.192	2.352	3	37	14	13	
15	153	26	26	26	49	49	49	2.180	2.180	.760	4.592	4.592	1.752	4	49	48	15	14
16	163	15	15	15	15	15	15	2.370	2.370	.950	5.352	5.352	2.512	2	16	15		
17	173	41	41	16	36	36	16	2.380	2.380	1.140	5.780	5.780	3.272	3	36	17	16	
18	183	46	46	17	61	61	17	2.180	2.180	1.360	4.980	4.980	4.152	3	61	18	17	
19	193	6	6	6	5	5	5	2.140	2.140	1.780	4.820	4.820	4.180	2	18	5		
20	203	24	29	29	22	22	22	2.380	2.380	.960	5.392	5.392	2.552	4	38	37	22	21
21	213	42	42	38	39	39	30	2.260	2.260	1.110	5.300	5.300	3.152	4	39	38	30	29
22	223	7	7	7	42	42	42	2.050	2.050	1.690	4.460	4.460	3.740	3	53	42	41	
23	233	6	6	6	45	45	45	1.960	1.960	1.600	4.100	4.100	3.380	4	63	62	46	45
24	243	23	23	25	46	46	47	2.185	2.185	.960	5.000	5.000	2.552	4	47	46	33	32
25	253	15	15	15	48	48	48	2.230	2.230	.810	4.792	4.792	1.952	4	48	47	25	24
26	263	113	113	113	72	72	72	2.140	2.140	.720	4.432	4.432	1.592	3	73	72	49	
27	273	28	28	28	20	20	20	2.480	2.480	1.060	5.792	5.792	2.952	3	50	20	19	
28	283	20	20	20	21	21	21	2.430	2.430	1.010	5.592	5.592	2.752	3	52	21	20	
29	293	30	30	30	23	23	23	2.330	2.330	.910	5.192	5.192	2.352	3	57	23	22	
30	303	25	25	25	24	24	24	2.280	2.280	.860	4.992	4.992	2.152	3	56	24	23	
31	313	25	25	25	25	25	25	2.280	2.280	.860	4.992	4.992	2.152	3	65	26	25	
32	323	31	31	31	26	26	26	2.330	2.330	.910	5.192	5.192	2.352	3	67	27	26	
33	333	32	32	32	27	27	27	2.380	2.380	.960	5.392	5.392	2.552	2	69	27		
34	363	9	9	28	51	51	52	2.300	2.300	1.160	5.460	5.460	3.352	4	52	51	29	28
35	373	23	23	36	62	62	54	2.010	2.010	1.210	4.300	4.300	3.552	3	62	54	53	
36	353	35	35	37	54	54	55	2.100	2.100	1.120	4.660	4.660	3.192	3	58	55	54	
37	343	36	36	24	55	55	32	2.210	2.210	1.010	5.100	5.100	2.752	4	56	55	32	31
38	403	36	36	37	58	58	31	2.220	2.220	1.060	5.140	5.140	2.952	4	58	57	31	30
39	413	44	44	24	64	64	33	2.230	2.230	1.010	5.180	5.180	2.752	4	65	64	34	33
40	423	39	39	39	34	34	34	2.280	2.280	1.060	5.380	5.380	2.952	4	67	66	35	34
41	433	40	40	40	35	35	35	2.330	2.330	1.110	5.580	5.580	3.152	4	69	68	36	35
42	453	43	43	21	40	40	39	2.150	2.150	1.220	4.860	4.860	3.592	3	43	40	39	
43	473	22	22	42	41	41	40	2.120	2.120	1.250	4.740	4.740	3.712	3	43	41	40	
44	493	23	23	39	63	63	64	2.020	2.020	1.220	4.340	4.340	3.592	3	64	63	59	
45	503	44	44	40	59	59	60	2.090	2.090	1.280	4.620	4.620	3.832	3	66	60	59	
46	513	45	45	45	60	60	60	2.140	2.140	1.330	4.820	4.820	4.032	3	68	61	60	
47	523	48	48	48	71	71	71	2.870	2.870	1.450	6.033	6.033	3.193	2	71	70		
48	533	121	121	121	74	74	74	2.650	2.650	1.230	5.656	5.656	2.816	4	169	75	74	71
49	543	48	48	48	75	75	75	2.700	2.700	1.280	5.776	5.776	2.936	3	81	76	75	
50	545	49	49	49	76	76	76	2.740	2.740	1.320	5.872	5.872	3.032	3	159	77	76	
51	553	49	49	49	81	81	81	2.740	2.740	1.320	5.936	5.936	3.096	2	82	81		
52	563	54	54	54	83	83	83	2.830	2.830	1.410	6.152	6.152	3.312	4	85	84	83	82
53	573	52	52	52	84	84	84	2.890	2.890	1.470	6.392	6.392	3.552	2	85	84		
54	583	50	50	50	77	77	77	2.790	2.790	1.370	5.992	5.992	3.152	3	83	78	77	
55	593	54	54	54	78	78	78	2.850	2.850	1.430	6.136	6.136	3.296	3	118	79	78	
56	603	55	55	55	79	79	79	2.950	2.950	1.530	6.376	6.376	3.536	3	116	80	79	
57	613	56	56	56	80	80	80	3.040	3.040	1.620	6.592	6.592	3.752	3	112	111	80	
58	623	57	57	57	111	111	111	3.070	3.070	1.650	6.712	6.712	3.872	4	115	114	111	110
59	633	58	58	58	110	110	110	3.130	3.130	1.710	6.952	6.952	4.112	2	114	110		
60	643	57	57	57	112	112	112	3.070	3.070	1.650	6.712	6.712	3.872	3	115	113	112	

```
GOOD ENTRY/EXIT NODES FOR SECTION  1

FROM DUMP      50
FROM GARAGE   240   90  176
  TO DUMP      50

GOOD ENTRY/EXIT NODES FOR SECTION  2

FROM DUMP     480  200  240  390  400
FROM GARAGE   200
  TO DUMP     480  260  240  390  400

GOOD ENTRY/EXIT NODES FOR SECTION  3

FROM DUMP     150  360   90  210
FROM GARAGE   150
  TO DUMP     150  340   90  210

GOOD ENTRY/EXIT NODES FOR SECTION  4

FROM DUMP      60
FROM GARAGE   390  530   40  360
  TO DUMP      60

GOOD ENTRY/EXIT NODES FOR SECTION  5

FROM DUMP     600  670
FROM GARAGE   770  670  640  760  730
  TO DUMP     670  600

GOOD ENTRY/EXIT NODES FOR SECTION  6

FROM DUMP    1090 1070
FROM GARAGE  1090 1670
  TO DUMP    1090 1070

GOOD ENTRY/EXIT NODES FOR SECTION  7

FROM DUMP    1260 1420
FROM GARAGE  1260 1420
  TO DUMP    1260 1420

GOOD ENTRY/EXIT NODES FOR SECTION  8

FROM DUMP    1320 1640 1560
FROM GARAGE  1320 1640 1560
  TO DUMP    1320 1640 1560
```

GLOSSARY

Air Force Refuse-Collection Scheduling Program: a set of four computer programs that perform residential refuse-collection scheduling and produce printed schedules and maps of the routes.

binary tree: a data structure in which each element can be linked to at most two other elements, referred to as left and right branches.

entry node: a node in a collection region at which a path from the garage or landfill reaches the section.

exit node: a node in a collection region at which a path to the landfill starts.

node: a numbered point on a street at which some characteristic of the street changes.

odd node: a node at which an odd number of segments meet.

profitable cycle: a cyclical path of paired odd nodes that reduces the total pairing length if new pairs are formed from the consecutive unpaired nodes.

section: a group of segments serviced by the same collection-vehicle trip.

segment: a portion of a street between two nodes.

(The reverse of this page is blank.)

INITIAL DISTRIBUTION

ADTC/CS	1
DDC/DDA	2
HQ AFSC/DL	2
HQ USAF/RDPS	1
AFIT/Library	1
AFIT/DE	1
AFIT/LSGM	1
National Science Foundation	1
EPA/ORD	1
USA-CERL/EH	1
USA Chief, R&D/EQ	1
USN Chief, R&D/EQ	1
AFETO/DEV	1
Hq AUL/LSE 71-249	1
Det 1 ADTC/TST	1
Det 1 ADTC/ECW	3
Det 1 ADTC/EC	1
USA-CERL/Library	1
USA-CERL	1
UNM-CERF	3

(The reverse of this page is blank)